GRASS-ROOTS

microservices

(I had some and I didn’t know)
What’s that?

- Overall system split into parts

- Each part focuses on a “single” business service
  - The actual meaning of single depends on the context

- Single Responsibility Principle applied to architecture
What is this?
MANY TIMES, THINGS LOOK DIFFERENT FROM WHAT THEY ARE…

We should really consider microservices…

DON’T BE SILLY! GO TO WORK !!!
A particular way of designing software applications as suites of independently deployable services.

- Run in dedicated process
- Communicate through lightweight mechanisms
- Simplified and automated deployment
- Decentralized control of languages and data

https://martinfowler.com/articles/microservices.html
IN OTHER WORDS …

SOA
JUST MADE SIMPLER
Tenet 1: Boundaries are explicit.
Tenet 2: Services are autonomous.
Tenet 3: Services share schema & contract, not class.
Tenet 4: Service compatibility is based upon policy.
Wasn’t this just SOA?

- **Service-oriented Architecture**
  - Primarily about reuse of business services
  - Coarse-grained and made of conceptually bigger components
  - Purpose of making the architecture more efficient and scalable

- **Microservices**
  - Primarily about team organization and replaceability of functions
  - Fine-grained, devisable as internal components of large business services
  - Purpose of making single pieces easier to replace and scale
Perspectives of Scalability

▪ **SOA**
  ▪ We build an overall scalable architecture

▪ **Microservices**
  ▪ We create independently scalable smaller components and build an overall scalable architecture by harmonizing the scalability of constituent blocks
MONOLITHIC APPLICATION
SEGMENTED APPLICATION
Challenges of Modern Software

- **Easy to replace**
  - Rewrite entire sections from scratch to add functions and/or switch to a different stack of technologies

- **Easy to deploy**
  - Frequent releases require the certainty of being able to deploy quickly and reliably without side effects and regression
  - Update individual blocks as soon as they get ready

- **Easy to scale**
  - Scalability at a finer level of granularity
At a Glance

- Independent vertical stacks of logic and data
- Independently packaged and deployed

- Communicating (if necessary) in some way
  - Mostly and preferably via API
“Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations.”

Melvin Conway, 1967
Teams and Microservices

- One team gets one (family of) microservice(s)
  - Actual size and granularity always depend on the project and organization

- Bounded Context in DDD
  - Theoretical foundation of microservices

- Common implementation
  - One team responsible for global technology stack
  - Each team free to make decisions and select tools and technologies
  - High autonomy, little coupling with other teams
DOMAIN-DRIVEN DESIGN

Strategic Design
- Ubiquitous language
- Bounded contexts

Implementation
- Domain model
- Layered architecture
Bounded Context

- Ubiquitous language
- Independent implementation
- External interface (to other contexts)
Why Having Bounded Contexts

- Functional areas of the application that are better treated separately
- Same term meaning different things to different people
- Same term used to indicate different elements
- Dependency on external subsystems
- Dependency on legacy code
Mapping Processes to Microservices
Independence of Microservices

- **Lifecycle**
  - Each microservice has its own implementation and stack
  - Can (ideally) be deployed without synchronization with other teams

- **Contracted interface**
  - Stable interfaces for communicating
  - Versioning

- **Data storage**
  - Owner of its data, data duplication is not an issue
  - May keep local copies of global data
Interaction between Microservices

- **Communication**
  - Connected microservices should expect communication to be async
  - Critical microservices should be isolated in multiple pools so that if one fails operations can continue

- **Robustness**
  - Build microservices to be resilient to errors and always try to recover
  - Assume you absorb in some way the failure of connected services

- **Functional scalability**
  - Each microservice takes the resources it needs to work as expected
  - Independent scalability
When You Get Microservices (and you didn’t know)
The reason we ended up to microservices has to do much more with productivity than pure technical matters or technologies.

It had to do much more with common sense and attitude to solve concrete (and small) business problems than religion or design for big-scale.
Every method is a Microservice

```java
for (Double amount : amounts) {
    result.add(amount + dest);
}
return result;
```
Humphrey’s Law
The user of the software won’t know what she wants until she sees the software.

Wegner’s Lemma
An interactive system can never be fully specified nor can it ever be fully tested.

Pluralsight courses
UXDD/DDD