Week 6: Additional Topics

Unit 6: Microservice Design – First Advice

Run Simple

openSAP
open.sap.com

Microservice Design – First Advice

Questions addressed in this unit

So far, we've focused on writing *one* microservice well.

Some additional key design topics for cloud-native applications are:

- How to split functionality into microservices?
- How do services communicate with each other?
- How to design microservices internally?
- What is the scaling model?
- How to set up your development organization?
- Where to find resources and design guidance?

This section gives *initial guidance* on how to start.

Microservice Design – First Advice

Disclaimer

Disclaimer!

This section does not claim to really explain the relevant topics of design, architecture, team setup, process etc.

The goal is just to give a rough idea and create awareness of the topic that you really need to work on in your project!

Microservice Design – First Advice

Microservice disasters

One example of how NOT to do it:

- Monolith broken down into **150** microservices
- > 100 developers, 18 months, everyone wrote ‘their service’
- One architecture guideline: No data duplication!
- Other guideline: REST APIs (synchronous!) for everything, huge number of internal API calls, no Hystrix used
- REST calls took on average 150 ms, target SLA 99.9%;
  200 calls needed to service one outside request => 13% SLA
- No continuous integration / delivery / automation infrastructure

For stories search youtube with "microservice disasters"

Microservice Design – First Advice

Introduction to Domain-Driven Design

DDD is a Design Method

- invented by Eric Evans for large system development / evolution, fits very well for cloud-native applications
- leads to highly cohesive and loosely coupled services

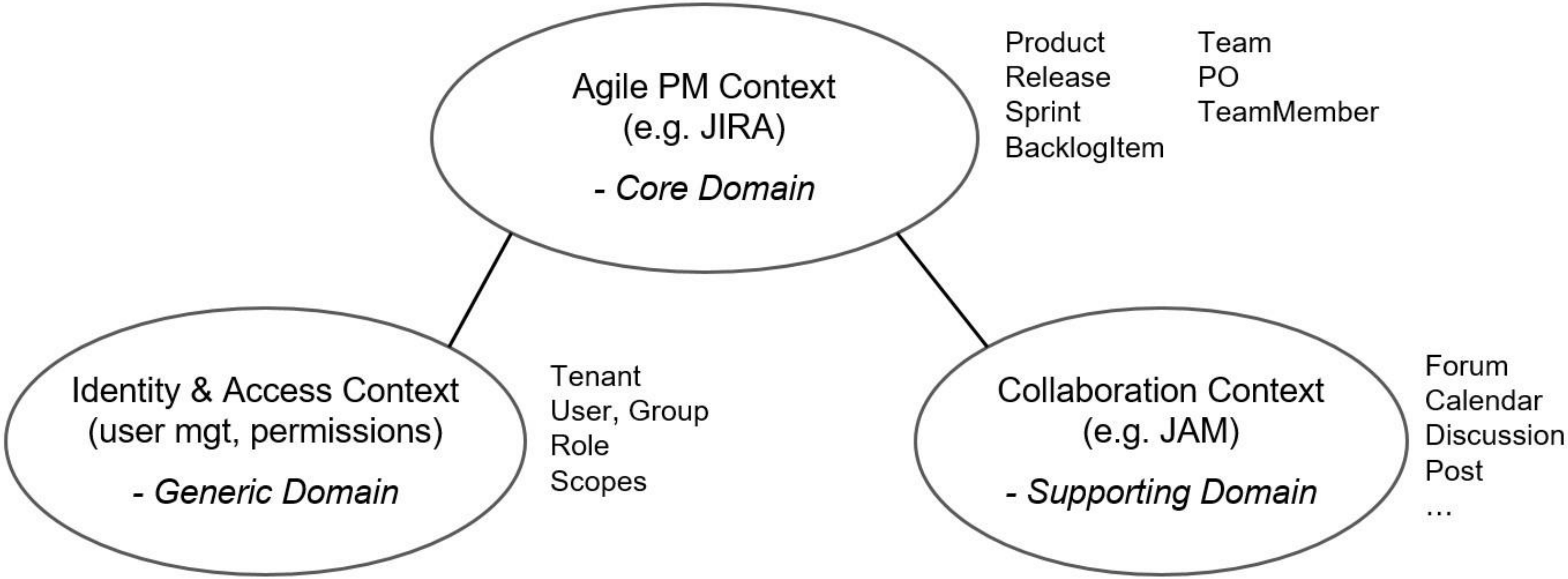
DDD – *Strategic Design* Level

- Observation: Large systems don't have "one complete and consistent model"
- Concerned with finding the domains and subdomains of your application
  - **Core domain**: Your area of innovation/IP
  - **Generic domain**: Stuff that many apps need, e.g. user management
  - **Supporting domain**: In-between, specific to your case
- Key: *Focus DDD design efforts on the core domain!*
- Buy generic, build core and supporting domain code

Microservice Design – First Advice

Context maps

A **context map** is useful to show the domains / bounded contexts, their types and relationships.

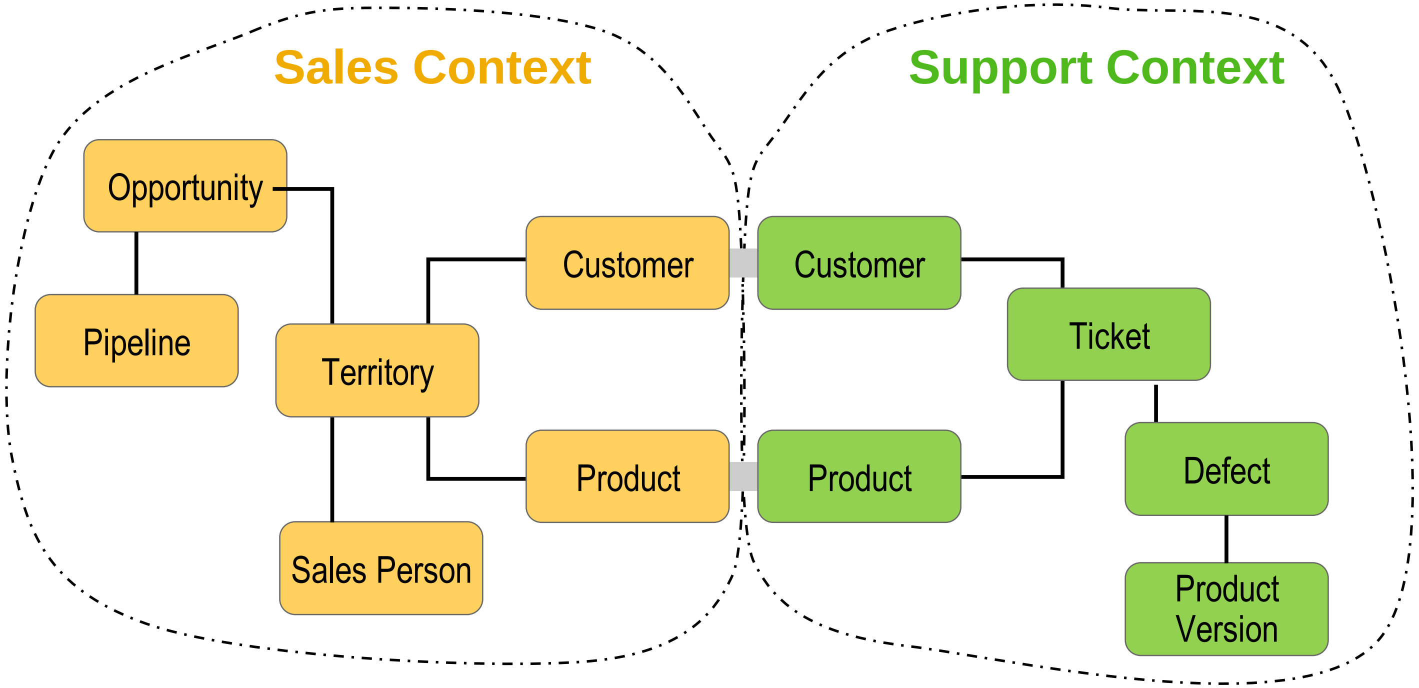


Microservice Design – First Advice

Model discovery and bounded contexts

Identify core concepts in the (sub-) domain by modeling key use cases.

- The same **term** usually means different things in different contexts. Model (and implement) them separately, i.e. with *separate data*
- A **bounded context** is a domain scope where concepts and language are logically complete and consistent.

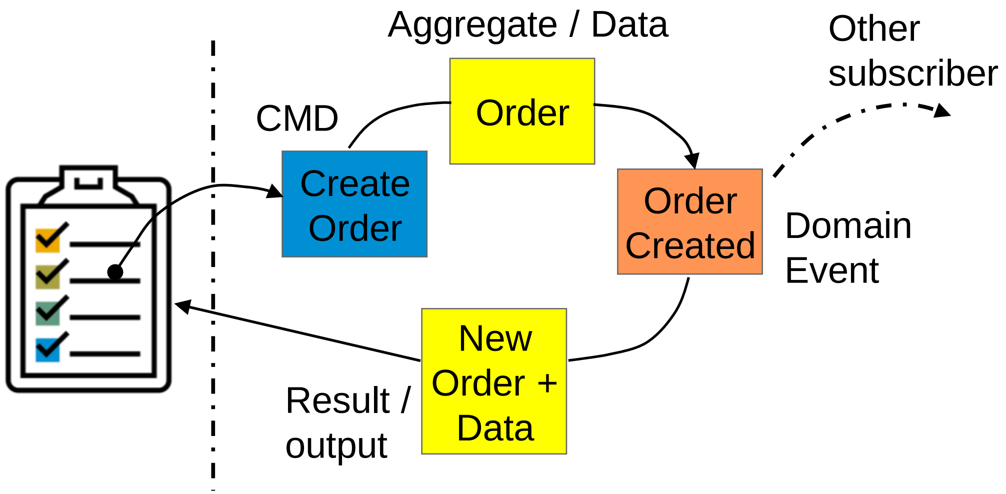


Microservice Design – First Advice

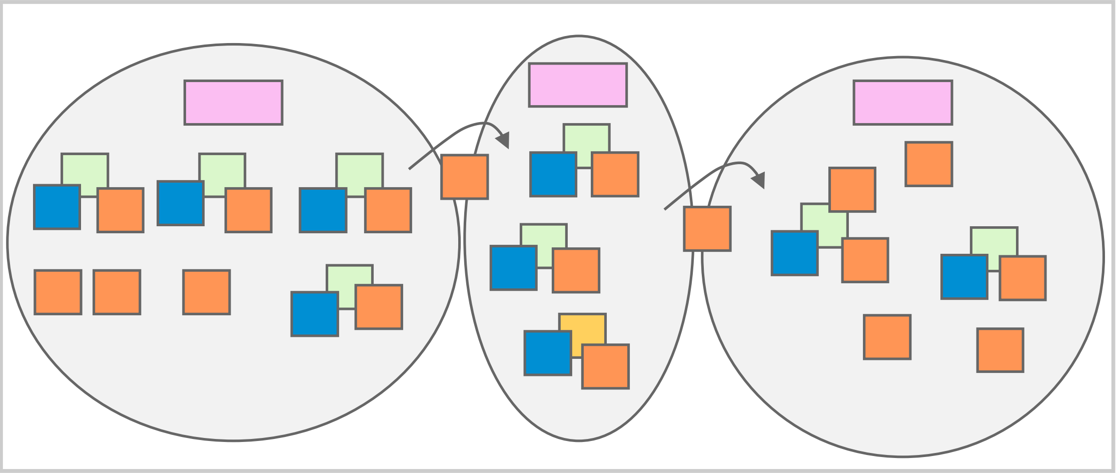
Event storming for business process discovery

Event Storming

- Outside-in domain modeling approach
- Often a moderated workshop with developers and domain experts
- ***Focus on business process, not on data!***



- Start with modeling process by relevant **business events**
- Identify needed data → Identify domains and sub-domains

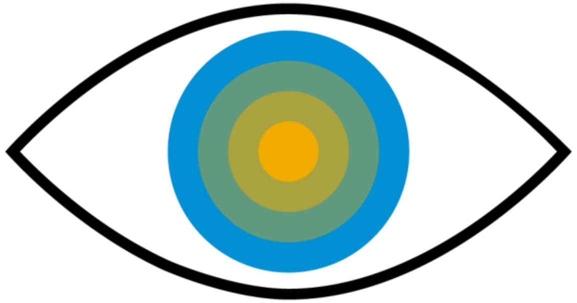


Microservice Design – First Advice

Key elements of DDD implementation design

- **Entity**: Has identity and attributes (e.g. Customer, Order)
- **Value Object (VO)**: Has NO identity (e.g. Address), favor VOs
- **Aggregate**: A few entities and VOs that have common lifetime, generally saved and loaded as one
- **Invariant**: Consistency rule that must be true, implemented at aggregate level, defines transaction boundary
- **Domain Event**: Something relevant happened in system, used within *and between* microservices for eventual consistency
- **Factories** create aggregates, **Repositories** manage persistence

Good aggregate design is a key ingredient to scalability.

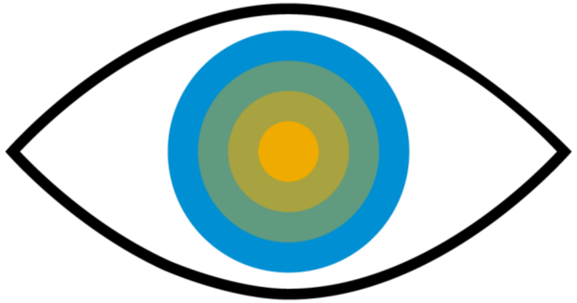


Microservice Design – First Advice

Making the right architecture / design choices

Remember the microservice architecture constraints:

- Each microservice has its own and separate data / DB
- No transaction across microservices; instead eventual consistency
- Calls between microservices are 10000x slower* than within
- Must design for failure everywhere (hardware, software)
- Scale horizontally
- Build in resilience to reach > 99.99%



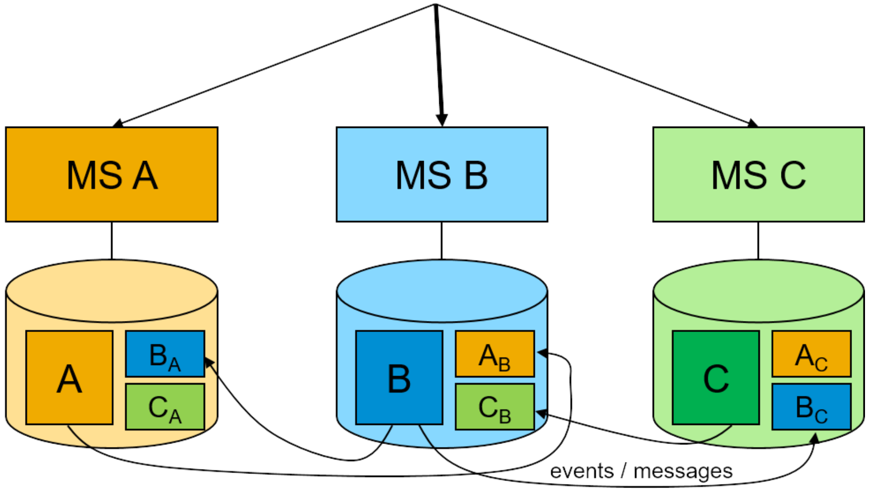
* REST call ≈ 10-20 ms; Java method call < 1μs

Microservice Design – First Advice

Initial design guidelines

First: Know your domain! Then start with these guidelines:

- Start with a few microservices, one microservice per bounded context
  – Separate out smaller microservices only when needed for scalability
- A microservice owns some data and replicates some data from others
  ⇒ can answer most requests without sync. calls to other microservices
- Use async. messages / events for updates between microservices



B_A: What A needs from B's data

Microservice Design – First Advice

Further reading

i Additional Material

- DDD eLearning by Eric Evans
- Splitting the monolith (Chapter 5 of ‘Building Microservices’)
- Cloud Design Patterns (Microsoft)
- Microservice Architecture of Otto.de
- Recommended Books:
  - Implementing Domain-Driven Design
    by Vaughn Vernon, publisher: Pearson Education
  - Building Microservices
    by Sam Newman, publisher: O'Reilly

Microservice Design – First Advice

How to get started

Cloud applications raise new and significant challenges

- Managers and architects: read “Building Microservices”
- Architects and developers: read “Implementing Domain-Driven Design” and/or other DDD books
- Go to a DDD training & apply it to your ‘core domain’
- Design with load and TCO in mind!
- Implement full test automation and continuous delivery
- Set up teams to own and operate 1-2 services

Thank you.

Contact information:

open@sap.com

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.