Tradeoffs, Prioritizing Advantages, & Levels

In this lesson, we'll look at some ways we can prioritize advantages of the microservice architecture and some potential trade-offs that should be considered.

WE'LL COVER THE FOLLOWING

Prioritizing advantages

Which of the discussed reasons for switching to microservices is the most important **depends on the individual scenario**. The use of microservices in a greenfield system is the one exception.

More often, a deployment monolith is replaced by a microservice system (see chapter 4). In that case, different advantages are relevant.

- The easier **scaling of development** can be an important reason for the introduction of microservices in such a scenario. Often, it is impossible to work quickly enough with a large number of developers on a single deployment monolith.
- The **easy migration** away from the legacy deployment monolith facilitates the introduction of microservices in such a scenario.
- **Continuous delivery** is often an additional goal. The aim is to increase the speed and reliability with which changes can be brought into production.

The scaling of development is not the only scenario for a migration. For example, when a **single Scrum team** wants to implement a system with

microservices, **scaling development would not be a sensible reason** since

the organization of development is not large enough for this. However, **other reasons are possible**. Continuous delivery, technical reasons like robustness, independent scaling, free technology choice, or sustainable development all play a role in such a scenario.

In the end, it is important to **focus on increasing the business value**. Depending on the scenario, an advantage in one of the previously mentioned areas might make the company more profitable or competitive, for example, faster time to market or better reliability of the system.

Microservices involve trade-offs

Depending on the aims, a team can compromise when implementing microservices.

- For example, when **robustness is the goal** of introducing microservices, the microservices have to be implemented as separate Docker containers.
  - Each Docker container can crash without affecting the other ones.
- **If robustness does not matter**, other alternatives can be considered. For example, multiple microservices can run together as Java web applications in one Java application server. In this case, they all run in one process and therefore are not isolated in respect to robustness. Still they are independently deployable.
  - A memory leak in any of the microservices will cause them all to fail.
  - However, such a solution is easier to operate and therefore might be the better trade-off in the end.

Two levels of microservices: Domain and technical

The technical and organizational advantages point to **two levels** at which a system can be divided into microservices.

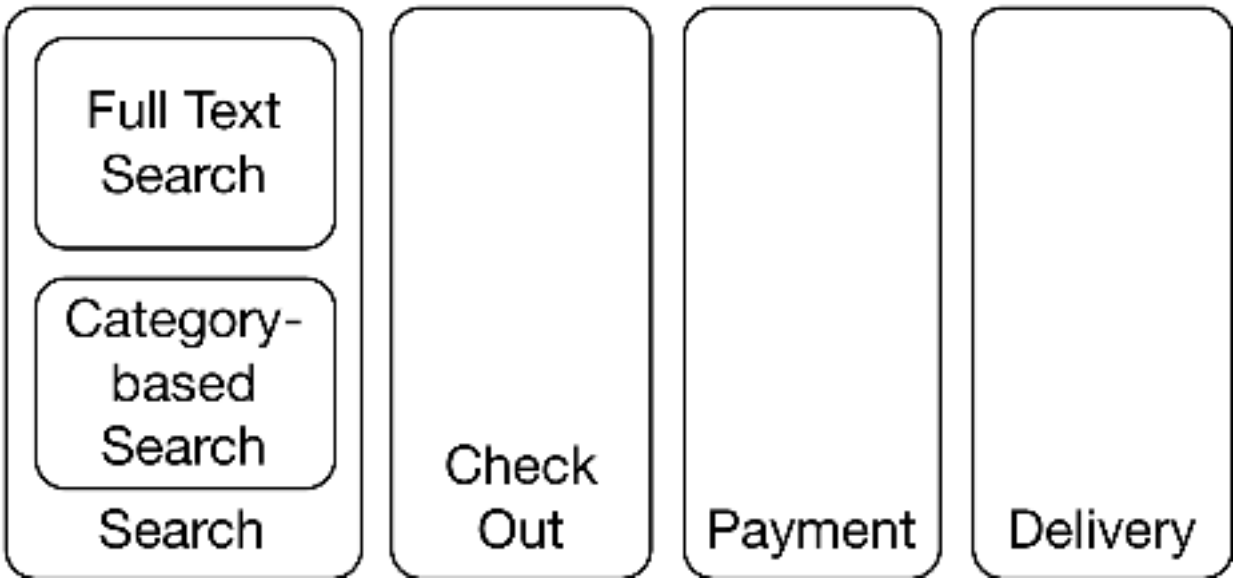
- A **coarse-grained division by domain** enables the teams to develop largely independently and allows them to roll out a new feature with the deployment of a single microservice if it concerns just this one domain, as it usually does. However, sometimes multiple domains are involved and

more than one microservice must be deployed.

    - For example, in an e-commerce system, customer registration and the order process can be examples of such coarse-grained microservices.

- For **technical reasons** some microservices can be *further divided*. These microservices can then be scaled independently of the other microservices.

    - When, for example, the last step of the order process is under especially high load, this last step can be implemented in a separate microservice. The microservice belongs to the domain of the order process, but for technical reasons, it is implemented as a separate microservice. This is an example of a **technical division**.



Two Levels of Microservices

The drawing above shows an example for the two levels. **Based on the domains**, an e-commerce application is **divided into** the microservices:

- Search
    - Full-text search
    - Category-based search
- Check out
- Payment
- Delivery

Search is further subdivided. The full-text search is separated from the category-based search.

- **Independent scaling** can be one reason for this. This architecture allows the system to scale the full-text search independently of the category-based search which is advantageous when both have to deal with different levels of load.
- Another reason could be the **use of different technologies**. The full-text search can be implemented with a full-text search engine, which is unsuitable for a category-based search.

Typical numbers of microservices in a system

It is difficult to state a typical number of microservices per system. Based on the divisions discussed in this chapter, 10-20 coarse-grained domains are usually defined, and each of these might be subdivided into one to three microservices. However, there are also systems with far more microservices.

QUIZ

1 A division by domain always results in the deployment of a single microservice for a change in the system.

COMPLETED 0%

1 of 3

In the next lesson, we'll look at some challenges that the microservice architecture poses.