Build, Operation, and Organization

In this lesson, we'll discuss the build, operation, and organization pertaining to moving a legacy system to a microservices architecture.

WE'LL COVER THE FOLLOWING

- Co-existence between microservices and legacy systems
- Integration test of microservices and legacy systems
- Coordinated deployment between legacy systems and microservices
- Organizational aspects
- Recommendation: do not implement all aspects at once

Code migration alone is not enough to turn a legacy system into a microservices system.

- **The microservices must also be built**. A suitable tool must be selected for this purpose. In addition, the continuous integration server has to cope with the multitude of microservices.
- Similarly, technologies and approaches must be introduced to **enable the deployment and operation** of microservices.
- Finally, **a suitable test strategy must be established**. This also requires the automated setup of test environments and the assurance that the tests are independent.
    - For example, stubs that simulate microservices or the legacy system are useful for this purpose, as are consumer-driven contract tests. They safeguard the requirements for the interfaces of microservices or legacy systems with the help of tests.
    - However, legacy systems are often very complicated, so these techniques are difficult to implement.

Therefore, dealing with the first microservice can require extra effort because the infrastructure for build and deployment needs to be set up. It is conceivable to build the infrastructure later, but it is recommended to start building the infrastructure as early as possible in order to reduce the risk of migration.

One or a few microservices can still be operated with an inadequate solution for build and deployment. However, once the number of microservices increases, without appropriate infrastructure, the necessary effort will become so high that it can lead to project failure.

Co-existence between microservices and legacy systems

During migration, the legacy system must be deployed and further developed in addition to the microservices. It is unrealistic to deploy the legacy system as often as the microservices because the effort of deploying the legacy system is usually far too high.

Therefore, **changes affecting both the legacy system and the microservices are difficult to implement**. They require at least one deployment of the microservices and one deployment of the legacy system. Solutions can be found at the architectural level.

If a new feature is implemented only in a microservice, then the deployment of only this microservice is necessary. This speaks for a division of the microservices according to bounded context.

Another option would be to **integrate the monolith with patterns** such as *open host service* or *published language*, as described in the previous chapter, to provide a generic interface that rarely needs to be changed.

Integration test of microservices and legacy systems

There must also be integration tests that test microservices with the version of the legacy system currently in production and with the one currently being developed. This ensures that the microservices continue to work when the legacy system is deployed.

The legacy system can support two different versions of the interfaces, so that microservices can switch to a new version of an interface when it is provided. However, no microservice is forced to use a new interface that has not yet been tested together with the microservice. In this way, the version of the microservice that uses a new interface of the legacy system can be deployed at any time.

Coordinated deployment between legacy systems and microservices

Coordinated deployment of microservices together with the legacy system is an alternative. When a change is made, the new version of **the microservices and the legacy system are rolled out at the same time**. However, this approach has a few disadvantages:

- This increases the risk because more changes occur at the same time
- It is harder to roll back the deployment.
- It is also difficult to implement this approach without downtime.
- With a complex microservices environment, this option is hardly possible anymore because too many microservices would have to be deployed at once.

Therefore, the deployment of microservices and legacy systems should be decoupled from the outset.

Organizational aspects

An essential advantage of microservices is the possibility to scale the development process.

If the goal of migration to microservices is to have independent teams, the migration of the architecture must be accompanied by a reorganization. This lesson in the previous chapter discusses the essential aspects of the target organization.

The organizational change must be coordinated with the technical migration. For example, a microservice can be detached from the legacy system and then

developed autonomously by a team. At the same time, the other

organizational structures can be set up – for example, the ones required for defining the macro architecture.

Recommendation: do not implement all aspects at once

Microservices require changes in architecture and organization as well as the introduction of new technologies.

Implementing all these changes at once is risky and complicated. Unfortunately, many of the changes are connected. Without new technologies, the architecture is difficult to implement. Without the architecture, organizational changes are difficult to introduce.

However, making all these changes at once should still be avoided. For each change, the question as to whether the change is actually necessary should be asked, in order to implement it at a later point in time.

QUIZ

1 Why are changes that affect both the legacy system and the microservices difficult to implement?

You can pick more than one answer.

In the next lesson, we'll discuss a few variations on the approaches discussed in this chapter.