# Alternative Strategies

In this lesson, we'll study a few alternative migration strategies.

WE'LL COVER THE FOLLOWING



There are many more migration strategies. There is a presentation that gives a good overview of the different approaches to the migration to microservices. Let's go over some of the most common approaches.

## Goal: reliability

As mentioned previously, there can be very divergent approaches for the migration to microservices. The strategy depends mainly on the objectives to be achieved.

When the main objective for switching to microservices is **an increase in robustness**, at first, **reliability can be improved at the interfaces to external systems** or databases with libraries like Hystrix and Resilience4j.

Then, **the system can be split step by step into individual microservices that run independently** of each other so that a failure of one microservice no longer affects the other microservices. There is an interesting talk about this approach to which slides are available.

## Migration based on layers

Another alternative is a **migration based on layers**.

For example, the UI can be migrated first. This can make sense when changes

For example, the UI can be migrated first. This can make sense when changes to the UI are imminent, and therefore the migration can be combined with

necessary changes.

Of course, this migration strategy is in contrast to the idea of combining UI, logic, and data in one self-contained system.

However, it can still be the first step towards this goal. In that case, **the remaining layers would have to be migrated into the same microservice afterward**. Alternatively, one stays with a division of microservices in layers, although it is not optimal. An ideal architecture, however, into which it is impossible to migrate, is a lot less helpful than a less optimal architecture that can actually be implemented.

## Copy/change #

Another possibility is copy/change. Here, the code of the **legacy system is copied**.

- In one copy, **one part of the system is developed further, while the other part is removed**.
- In a second copy, **it is the other way around**.

In this manner, the legacy system is converted into **two microservices**.

This approach has the **advantage** that the **old code is still used**, and therefore the functionalities of the microservices very likely correspond accurately to the functionalities of the legacy system.

### Caveats #

- However, **at the same time it is a great disadvantage to continue using the old code**. In most cases, the code of a legacy system is hard to maintain, and it is problematic to keep using this code.

- In addition, the database schema remains unaltered. The **shared use of the database schema by the legacy system and the microservices results in a tight coupling between the two**, which really should be avoided to be able to profit from the advantages microservices offer. That is why a black box migration might be better.

- Moreover, the structure of the legacy system and the technology stack largely remain the same. Thus, the project has a lot of **technical debt from the very beginning** and does **not represent a new start**.
- This approach does not take advantage of the benefits of microservices such as freedom of technology.

Therefore, **it should only be used in exceptional cases**.

## QUIZ

1 When should the copy/change strategy be used?

COMPLETED 0%

1 of 2

In the next lesson, we'll study the build, operation, and organization concerned with turning a legacy system into a microservices system.