Variations

In this lesson, we'll discuss some variations to the approaches discussed in this chapter.

WE'LL COVER THE FOLLOWING

In the domain macro architecture, strategic design and domain-driven design are ultimately unrivaled as approaches.

However, the bounded contexts depend on the specific project. **Identifying the right bounded contexts is a central challenge** when designing the architecture of a microservices system.

The **technical micro and macro architecture also have to be devised** for each project. This depends on many **factors**:

- Organizational aspects such as *DevOps organization* or having a separate operations team has an influence.
- In addition, *strategic technology decisions* can play a role.
- Even the *hiring policy* can be a factor. Eventually, there have to be experts available who can work in the teams to manage the technologies.

More complex rules

In reality, the rules of micro and macro architecture are **often more complex**.

For example, **a whitelist can exist for the programming language**. In addition, there can be a procedure for adding more programming languages to the whitelist, for example, via a committee. And finally, there can be a general limitation to programming languages that run on the JVM (Java

Virtual Machine).

Such a rule **has elements of a macro architecture decision**. There is a whitelist and a restriction to JVM languages. **At the same time, it also has micro architecture elements**. After all, a team can select one of the programming languages from the whitelist and even extend the whitelist.

Therefore, rules are often in place for every point in practice that allow the teams and microservices a certain amount of leeway. These rules are **not purely micro or macro architecture rules** but lie somewhere in between.

Experiments

The approach for defining micro and macro architecture can look like this:

- Consider a project you are familiar with. Look at its domain model and **consider the following questions**:
    - Would a division into multiple domain models and bounded contexts make the system easier?
    - In how many bounded contexts would you split the system? Typical projects consist of about ten bounded contexts. However, the exact number will vary for each individual project.
    - Determine the use cases which the system implements. Group use cases and analyze whether these use cases can be addressed by a domain model. By doing so, these use cases form a bounded context in which the domain model is valid.
    - Is a further division for technical reasons sensible? The technical reasons can comprise independent scalability or security (see also “Two Levels of Microservices” in this lesson).
- **Define for your project whether the individual decision should be part of micro or macro architecture**.
- **Work out at least one of the decisions in more detail**. For example, there could be a whitelist of programming languages, or only one programming language might be allowed that can be used by all microservices, or even a procedure for extending the whitelist.

QUIZ

Q

Suppose a whitelist for the databases is defined at the macro level. Which of the following is true concerning the decision to use a specific database?

COMPLETED 0%

1 of 1

In the next lesson, we'll look at the chapter conclusion.