Data Replication, Bounded Contexts, & Protocols

In this lesson, we'll study data replication, bounded contexts, and protocols.

WE'LL COVER THE FOLLOWING

Data replication and bounded context

Asynchronous communication becomes more complicated **if data is required** to execute a request.

For example, in the *catalog*, the *order process*, and the *invoice* data about products and customers has to be available.

Each of the systems stores a part of the information about these business objects.

- The *catalog* must display the products, so it has pictures and descriptions of the products.
- For *invoices*, prices and tax rates are important.

This corresponds to the definition of bounded contexts.

> Each **bounded context** has its own domain model. That means that all data for the bounded context is represented in its domain model. Therefore, the data specific for the bounded context should be stored in the bounded context in its own database schema.

Other bounded contexts should not access that data directly, which would

compromise encapsulation. Instead, the data should be accessed only by the logic in the bounded context and its interface.

Although it would be possible to have a system that contains all information about, for example, a product, this would not make a lot of sense. The model of the system would be very complicated.Also, it means that the domain model would be split across one system for an order process and another system for the product data. That **would lead to a very tight coupling**.

A third system, such as *registration* for customer data or *listing* for product data, must accept all the data and transfer the needed parts of the data to the respective systems. This can also be done asynchronously.

The other bounded contexts then store the information about products and customers in their local databases, making replication a result of events being processed. An event such as “new product added” makes each bounded context add some data to its domain model.

Registration or *listing* do not need to store the data. After they have sent the data to the other microservices, their job is done.

It is also possible to do an extract-transform-load approach. In that case, a batch would *extract* the data from one bounded context, transform it into a different format, and load it into the other bounded context. This is useful if a bounded context should be loaded with an initial set of data, or if inconsistencies in the data require a fresh start.

Synchronous communication protocols

Asynchronous communication as previously defined does not make any assumptions about the communication protocol used.
For **synchronous** communication, the server **must respond to each request**.

Examples are REST and HTTP. A request leads to a response that contains a status code and optional additional data. It is possible to implement asynchronous communication with a synchronous communication protocol. Chapter 8 explains this in more detail.

Asynchronous communication protocols

It is more natural to implement asynchronous communication with an

asynchronous communication protocol. An asynchronous communication

protocol **sends messages and does not expect responses**. Messaging systems like Kafka (see chapter 7) implement this approach.

There is also a presentation which explains the difference between REST and messaging, and highlights that both technologies can be used to implement synchronous communication like **request/reply**, but also asynchronous communication like **fire & forget** or **events**.

QUIZ

1 If a system is a bounded context it _____.

COMPLETED 0%

1 of 3

In the next lesson, we'll discuss events in asynchronous communication.