Benefits & Challenges

In this lesson, we'll discuss the benefits and challenges of the synchronous microservices approach.

WE'LL COVER THE FOLLOWING

Benefits

In practice, synchronous microservices are a relatively frequently used approach. Many well-known examples of microservice architectures use such a concept.

This has the following advantages:

- All services can access the same dataset, so there are **fewer consistency problems**.
- Synchronous communication is a **natural approach if the system is to offer an API**. Any microservice can implement part of the API. The API can be the product or the API can be used by mobile applications.

- After all, it can be **easier to migrate into such an architecture**. For example, the current architecture can already have a division into different synchronous communication endpoints or teams can exist for each of the functionalities.
- Calling methods, procedures, or functions in a program is usually synchronous. **Developers are familiar with this model** so they can understand it more easily.

Challenges

Synchronous microservices create a number of challenges. Let's discuss a few:

Low performance

The communication with other microservices during request processing causes the latencies for responses of other microservices and the communication times via the network to add up.

Therefore, synchronous communication **can lead to a performance problem**. When a microservice reacts slowly, this can have **consequences for a plethora of other microservices**.

- In the example in the last lesson, the *catalog* uses two other microservices (*item* and *customer management*). Thus, the latencies of these three systems can add up.
- A request to the *catalog microservice* creates requests to the *customer management microservice* and to the *item microservice*.
- Only when all microservices have responded to the requests, the user gets to see the result.

Failure propogation

When a synchronous microservice calls a failed microservice, the **calling microservice may crash and the failure propagates**. This makes the system very vulnerable.

The vulnerability of the microservices and the additional waiting times can **prevent the reliable operation of microservices systems** with synchronous communication. These problems have to be solved when a microservices

system uses synchronous communication.

Higher level of dependency

In addition, synchronous communication can create a higher level of dependency in the domain logic. Asynchronous communication often focuses on events (see Events from chapter 6).

In this case, a microservice can decide how to react to events. In contrast, synchronous communication typically defines what a microservice is supposed to do.

- In the example, the **order process** would make the **invoice microservice** generate an invoice.
- With events, the **invoice microservice** would decide how to react to the event.
- This facilitates the extendibility of the system.

If the customer is supposed to be credited with bonus points for an order, there has to be an additional microservice reacting to the already existing event.

That event is probably already processed by more than one microservice, so one more receiver needs to be added. In a **synchronous architecture, an additional system has to be called**.

Technical solutions

For implementing a system of synchronous microservices some **technical solutions** are required. Let's discuss them.

Service discovery

The microservices have to know how they can communicate with other microservices. Usually, this requires an **IP address and a port**.

> **Service discovery** serves to find the port and IP address of a service.

Service discovery should be **dynamic**:

- Microservices can be **scaled**. Then there are **new IP addresses** at which additional instances of a microservice are available.
- In addition, a microservice can **fail**. Then it is **not available anymore** at the known IP address.

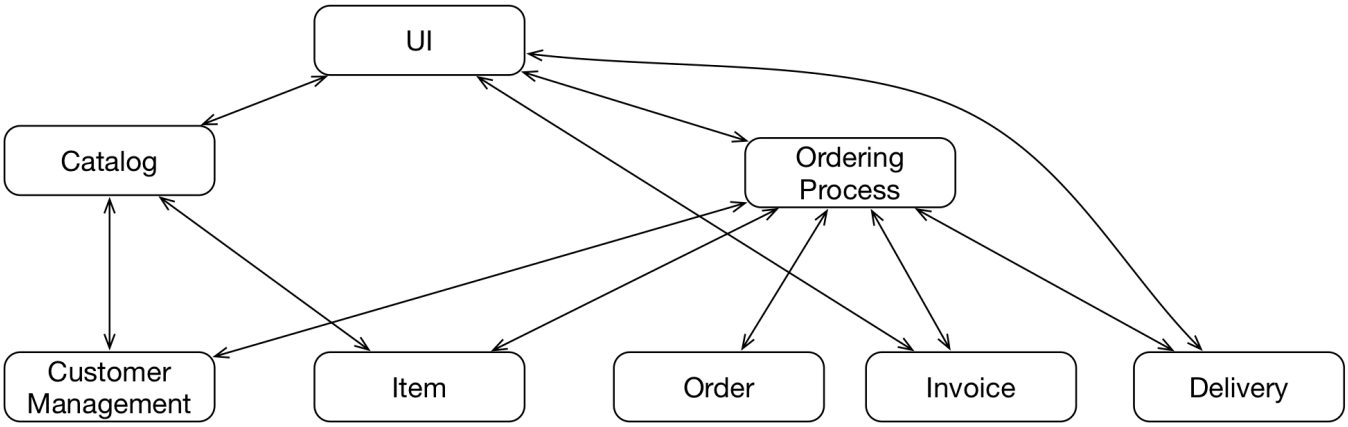
Service discovery can be very simple. **DNS (Domain Name System)** provides IP addresses for servers on the Internet.This technology already provides a simple service discovery.

Resilience

When communication is synchronous, microservices have to be **prepared for the failure of other microservices**. It has to be prevented that the calling microservice fails as well.

Otherwise, there will be **failure cascade**:

- First one microservice fails.
- Then other microservices call this microservice and fail.
- In the end, the entire system will be down.



When the service for the items fails in the architecture above, the catalog and order process could fail in turn.

This would render a large part of the system unusable. Therefore, there has to be a technical solution to achieve **resilience**.

Load balancing

Each microservice should be **scalable independently** of the other

microservices. Load has to be distributed between microservices.

This does not only pertain to access from the outside, but also to internal communication. Therefore, there has to be a *load balancing* for each microservice.

Routing

Finally, every access from the outside should be forwarded to the responsible microservices. This requires **routing**.

- A user might want to use the catalog and the order process.
- While they are separate microservices, they should appear as parts of the **same system** to the outside.

Consequently, the technologies for synchronous microservices which are discussed in the following chapters have to offer solutions for **service discovery, resilience, load balancing, and routing**.

API gateways

For complex APIs, complex routing of requests to the microservices might be needed. **API Gateways** offer additional features:

- Most of the API gateways can perform user authentication.
- They can throttle the network traffic for individual users to support a high number of users at the same time. This can be supplemented by centralized logging of all requests or caching.
- API gateways can also solve aspects like monitoring, documentation, or mocking.

The implementations of Apigee, 3scale by Red Hat, apiman, or cloud products like the Amazon API Gateway and the Microsoft Azure API Gateway are examples of API gateways.

The examples in this course do not offer public REST interfaces, only REST interfaces for internal use. The public interfaces are just web pages and the examples do not use API gateways.

QUIZ

QUIZ

1

What would happen if a system had hardcoded service discovery so that an IP address was written against a microservice in a file that all microservices had access to?

Choose two options from the answers below.

COMPLETED 0%

1 of 2

In the next lesson, we'll look at some variations of the synchronous approach we've discussed already.