More on Advantages

In this lesson, we'll continue our discussion of the advantages of microservices.

Robustness

Microservice systems are more robust.

When a memory leak exists in a microservice, only this microservice is affected and crashes. The other microservices keep running. Of course, they have to compensate for the failure of the crashed microservice; this is called **resilience**.

To achieve resilience, microservices can cache values and use them in case of a problem. Alternatively, there might be a fallback to a simplified algorithm.

Without resilience, the **availability of a microservice system might be a problem**. It is likely that a microservice will fail for any reason.

- For example, due to the distribution into several processes, many more servers are involved in the system. Each of these servers can potentially fail.
- Communication between microservices occurs via the network, which can also fail. Therefore, microservices need to implement resilience to achieve robustness.

Independent scaling

Most of the time, **scaling the whole system is not required**. For example, for a shop system during Christmas, the catalog might be the most critical and hardware-consuming part. By scaling the complete system, the hardware is spent on parts that don't require more power.

Each microservice can be independently scaled. It is possible to start additional instances of a microservice and distribute the load of the microservice into the instances. This can improve the scalability of a system significantly.

So, in the previous example, just the catalog would need to be scaled up. For this to work, the microservices naturally have to fulfill **certain requirements**. For example, they must be stateless. Otherwise, requests of a specific client cannot be transferred to another instance, because this instance then would not have the state specific to that client.

It can be difficult to start more instances of a deployment monolith due to the required hardware. Besides, building up an environment for a deployment monolith can be complex. This can require additional services or a complex infrastructure with databases and additional software components.

In the case of a microservice, **the scaling can be more fine-grained** so that normally fewer additional services are necessary and the basic requirements are less complex.

Free technology choice

- Each microservice can be implemented with an individual technology. This facilitates the migration to a new technology since each microservice can be migrated individually.
- In addition, it is simpler and less risky to gain experience with new technologies since they can initially be used for only a single microservice before they are employed in several microservices.

Security

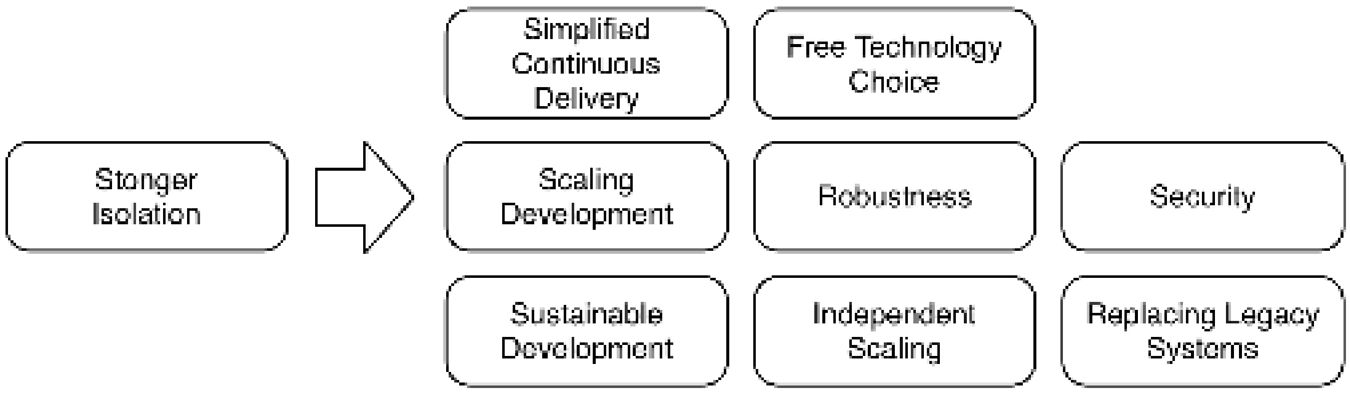
Microservices can be isolated from each other.

- For example, it is possible to **introduce firewalls** into the

- For example, it is possible to **introduce firewalls** into the communication between microservices.
- Besides, the **communication** between microservices **can be encrypted** to guarantee that the communication really originates from another microservice and is authentic. This prevents the corruption of additional microservices if a hacker takes over one microservice.

In general: isolation

In the end, many advantages of microservices can be traced back to a **stronger isolation**.



Isolation as the Source of the Advantages of Microservices

To sum it up:

- Microservices can be deployed in isolation, which **facilitates continuous delivery**.
- They are isolated in respect to failures, which **improves robustness**.
- The same is true for **scalability**. Each microservice can be scaled independently of the other microservices.
- The employed technologies can be chosen for each microservice in isolation, which allows for **free technology choice**.
- The microservices are isolated in such a way that they can only communicate via the network. Therefore, **communication can be safeguarded by firewalls**, which **increases security**.
- Due to this strong isolation, the boundaries between modules cannot be violated by mistake. The **architecture is rarely violated**; this safeguards the architecture

- In isolation, a microservice can be **replaced with a new microservice**. This enables the low-risk replacement of microservices and allows one to keep the architecture of the individual microservices clean. Thus, isolation facilitates the long-term maintainability of the software.
- **Decoupling** is an important feature of modules. With their isolation, microservices push it to the extremes. Modules are normally only decoupled in regard to code changes and architecture. The decoupling between microservices goes far beyond that. Thanks to decoupling, microservices are smaller. This serves many purposes:
    - Makes it easier to reason about them
    - The security of a microservice is easier to verify
    - The performance is easier to measure
    - It is easier to figure out whether they work correctly
    - That makes the design and also the development easier

QUIZ

1

In a microservice architecture, what will happen if one microservice crashes?

COMPLETED 0%

1 of 3

In the next lesson, we’ll look at a few trade-offs of using the microservice architecture, how to prioritize its advantages based on your application, the two levels of microservices and how many microservices can be expected per system.