In this lesson, we'll look at an overview of what to expect from this chapter!

**We'll cover the following**

* [Microservices: definition](https://www.educative.io/module/lesson/introduction-to-microservices/JYgqAp6oNY2#Microservices:-definition)
  + [Advantages of this microservice definition](https://www.educative.io/module/lesson/introduction-to-microservices/JYgqAp6oNY2#Advantages-of-this-microservice-definition)
  + [Deployment monolith](https://www.educative.io/module/lesson/introduction-to-microservices/JYgqAp6oNY2#Deployment-monolith)
  + [Size of a microservice](https://www.educative.io/module/lesson/introduction-to-microservices/JYgqAp6oNY2#Size-of-a-microservice)
* [Chapter walkthrough](https://www.educative.io/module/lesson/introduction-to-microservices/JYgqAp6oNY2#Chapter-walkthrough)

**Microservices: definition**[#](https://www.educative.io/module/lesson/introduction-to-microservices/JYgqAp6oNY2#Microservices:-definition)

Unfortunately, there is no universally acknowledged definition for the term *microservice*. In the context of this course the following definition will be used:

**Microservices** are independently deployable modules.

For example, an **e-commerce system** can be divided into modules for:

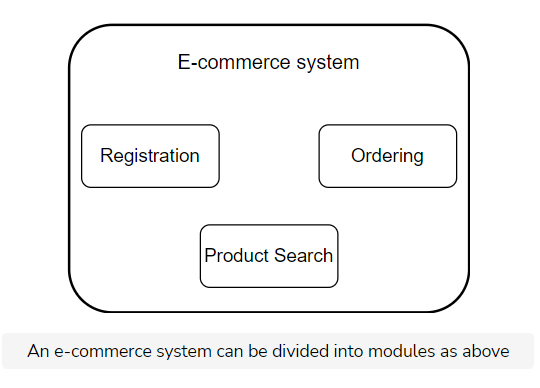
* ordering
* registration
* product search

Normally, all of these modules would be implemented together in one application. In this case, **a change in one of the modules** can only be brought into production by bringing a **new version of the entire application** with all its modules into production. However, when the modules are implemented as microservices, the ordering process cannot only be **changed independently of the other modules**, but it can even be brought into production independently.

This speeds up deployment and reduces the number of necessary tests since only a single module needs to be deployed. Due to this greater level of decoupling, a large project can turn into a number of smaller projects. Each project is in charge of an individual microservice.

To achieve this at the technical level, **every microservice has to be an independent process**. A better solution for decoupling microservices is to provide an independent virtual machine or Docker container for each microservice.

In that case, a deployment will replace the Docker container of an individual microservice with a new Docker container, which starts the new version and its direct requests. The other microservices will not be affected if such an approach is used.



### Advantages of this microservice definition [**#**](https://www.educative.io/module/lesson/introduction-to-microservices/JYgqAp6oNY2#Advantages-of-this-microservice-definition)

The definition of microservices as independently deployable modules has several advantages:

* It is very compact.
* It is very general and covers all kinds of systems which are commonly denoted as microservices.
* The definition is based on modules and is thus a well-understood concept. This allows us to adopt many ideas concerning modularization. This definition also highlights that microservices are part of a larger system and cannot function entirely on their own. Microservices have to be integrated with other microservices.
* The independent deployment is a feature that creates numerous [advantages](https://www.educative.io/collection/page/10370001/6518081205567488/4998953437233152) and is therefore very important. Thus, the definition, in spite of its brevity, explains what the most essential feature of a microservice really is.

### Deployment monolith [**#**](https://www.educative.io/module/lesson/introduction-to-microservices/JYgqAp6oNY2#Deployment-monolith)

**A system that is not made up of microservices** can only be deployed in its entirety. Therefore, it is called a deployment monolith. Of course, a deployment monolith can be divided into modules. The term deployment monolith does not make a statement about the internal structure of the system.

### Size of a microservice [**#**](https://www.educative.io/module/lesson/introduction-to-microservices/JYgqAp6oNY2#Size-of-a-microservice)

The above definition of microservices does not say anything about the size of a microservice. Of course, the term microservice suggests that especially small services are meant. However, in practice, **microservices can vary hugely in size**. Some microservices keep an entire team busy, while others comprise only a few hundred lines of code. Thus, the size of microservices is ill-suited to be part of the definition.

## Chapter walkthrough [#](https://www.educative.io/module/lesson/introduction-to-microservices/JYgqAp6oNY2#Chapter-walkthrough)

This chapter introduces microservices and discusses:

* [Advantages](https://www.educative.io/collection/page/10370001/6518081205567488/4998953437233152) and [disadvantages](https://www.educative.io/collection/page/10370001/6518081205567488/4532272759832576) of microservices to enable the reader to evaluate the applicability and usefulness of this architecture for a specific project.
* The discussion of benefits explains which problems microservices can solve and how this architecture can be adapted for different scenarios.
* The discussion of disadvantages illustrates where technical challenges and risks lie and how these can be addressed.
* Recognizing advantages and disadvantages is critical for technology and architecture decisions since those have to be aimed at maximizing benefits and reducing disadvantages.

A microservice should not be any longer than a few hundred lines of code.

False

Microservices are \_\_\_\_\_.

###### independently deployable modules

A key feature of the microservices architecture is that a new version of the entire application is deployed when a change is made to any one microservice.

###### False

# Advantages

There are a number of reasons why we should use microservices. Let's discuss them in this lesson.

**We'll cover the following**

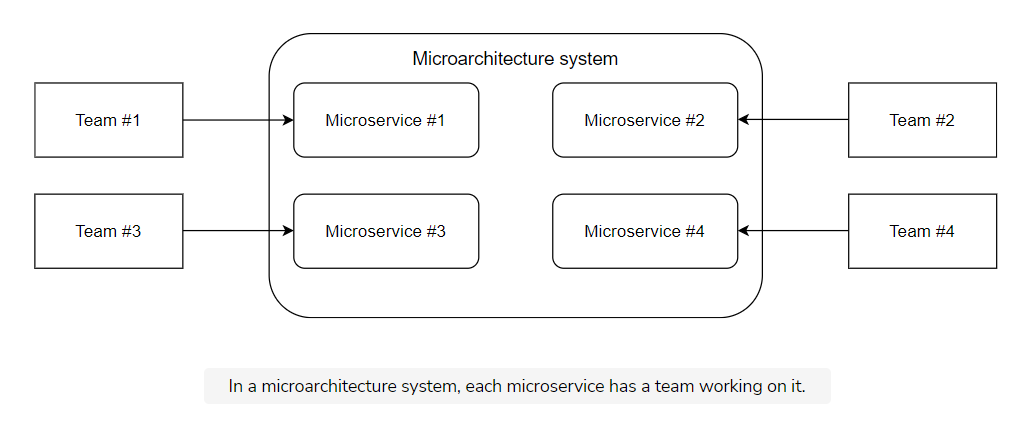
* [Microservices for scaling development](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#Microservices-for-scaling-development)
* [Replacing legacy systems](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#Replacing-legacy-systems)
* [Sustainable development](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#Sustainable-development)
  + [Replaceability of microservices](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#Replaceability-of-microservices)
  + [Dependencies have to be managed](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#Dependencies-have-to-be-managed)
    - [In Classical Architectures](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#In-Classical-Architectures)
    - [In the microservices architecture:](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#In-the-microservices-architecture:)

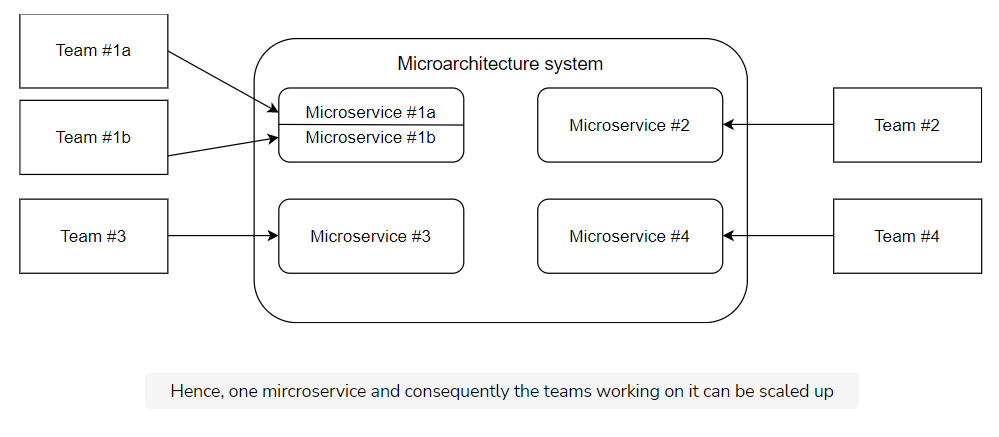
## Microservices for scaling development [#](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#Microservices-for-scaling-development)

One reason for the use of microservices is the **easy scalability of development**. Large teams often have to work together on complex projects. With the help of microservices, the projects can be divided into smaller units that can work independently of each other.

* For example, the **teams** responsible for an individual microservice **can make most technology decisions on their own**.
* When the microservices are delivered as Docker containers, **each Docker container only has to offer an interface** for the other containers.
* The **internal structure of the containers does not matter** as long as the interface is present and functions correctly. Therefore, it is irrelevant which programming language a microservice is written in. Consequently, the responsible team can make such decisions on their own. Of course, the selection of programming languages can be restricted in order to avoid increased complexity. However, even if the choice of the programming language in a project has been restricted, a team can still independently use an updated library with a bug fix for their microservice.
* As stated in the [last lesson](https://www.educative.io/collection/page/10370001/6518081205567488/6272204058656768), when a new feature only requires changes in one microservice, it **can not only be developed independently, but it can also be brought into production on its own**. This allows the teams to work on features completely independently.

Thus, with the help of microservices, **teams can act independently regarding domain logic and technology**. This **minimizes the coordination effort** required for large projects.





## Replacing legacy systems [#](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#Replacing-legacy-systems)

The maintenance of a legacy system is frequently a challenge since:

1. The code is often badly structured.
2. The changes are not checked by tests.
3. Developers might have to deal with outdated technologies.

Microservices help when working with legacy systems since the existing code does not necessarily have to be changed. Instead, **new microservices can replace parts of the old system**. This requires integration between the old system and the new microservices, for example, via data replication, REST, messaging, or at the level of UI. Besides, problems such as a uniform single sign-on for the old system and the new microservices have to be solved.

But then the microservices are very much like a [greenfield project](https://en.wikipedia.org/wiki/Greenfield_project). **No pre-existing codebase has to be used**. In addition, developers can employ a completely different technology stack. This immensely facilitates work compared to having to modify the legacy code itself.

## Sustainable development [#](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#Sustainable-development)

Microservice-based architectures promise that **systems remain maintainable** even in the long run.

### Replaceability of microservices [**#**](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#Replaceability-of-microservices)

An important reason for this is the **replaceability of microservices**. When a microservice can no longer be maintained, it can be rewritten. Compared to changing a deployment monolith, this entails less effort because the microservices are much smaller.

However, it is difficult to replace a microservice, on which numerous other microservices depend since changes might affect the other microservices. Thus, to achieve replaceability, **the dependencies between microservices have to be managed appropriately**.

Replaceability is a great strength of microservices. Many developers work on replacing legacy systems. However, when a new system is designed, the question of how to replace this system after it has turned into a legacy system is rarely asked. Microservices with their replaceability provide an answer.

Hence, individual microservices remain maintainable. If the code of a microservice is unmaintainable, it can just be replaced and it would not influence any of the other microservices.

### Dependencies have to be managed [**#**](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#Dependencies-have-to-be-managed)

To achieve maintainability, the dependencies between the microservices have to be managed in the long term.

#### In Classical Architectures [**#**](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#In-Classical-Architectures)

* **Classical architectures often have difficulties** at this level. A developer writes new code and unintentionally introduces a new dependency between two modules, which had been forbidden in the architecture.
* Typically, the mistake goes unnoticed because attention is only paid to the code level of the system and not to the architectural level.
* Often, it is not immediately clear which module a class belongs to. So it is also unclear to which module the developer just introduced a dependency.
* In this manner, more and more dependencies are introduced over time. The originally designed architecture becomes more violated, culminating in a completely unstructured system.

#### In the microservices architecture: [**#**](https://www.educative.io/module/lesson/introduction-to-microservices/g7mAkg83kM9#In-the-microservices-architecture:)

* Microservices have **clear boundaries due to their interface** irrespective of whether the interface is implemented as a REST interface or via messaging.
* When a developer introduces a new dependency on such an interface, they will notice this because the interface has to be called appropriately. For this reason, **it is unlikely that architecture violations will occur** at the level of dependencies between microservices.
* The interfaces between microservices are in a way **architecture firewalls** since they prevent architecture violations. The concept of architecture firewalls is also implemented by architecture management tools like [Sonargraph](https://www.hello2morrow.com/products/sonargraph" \t "_blank), [Structure101](http://structure101.com/), or [jQAssistant](https://jqassistant.org/" \t "_blank). Advanced module concepts can also, generate such a firewall. In the Java world, [OSGi](https://www.osgi.org/) limits access and visibility between modules. Access can even be restricted to individual packages or classes.

The architecture at the level of dependencies between microservices also remains maintainable. **Developers cannot unintentionally add dependencies** between microservices. Therefore, microservices can **ensure a high architecture quality** in the long term both inside each microservice and between the microservices.

Thus, microservices enable sustainable development where the speed of change does not decline over time.

Why is it NOT likely that a developer will introduce a new dependency between two modules in a microservice architecture?

###### C)Because microservices have clear boundaries due to their interface and to introduce a dependency, they will have to call it

# Advantage: Continuous Delivery

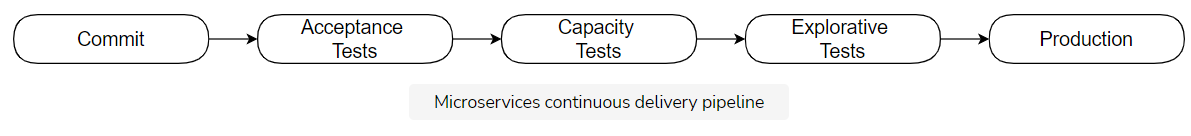
In this lesson, we'll focus on continuous delivery as an advantage of using microservices.

**We'll cover the following**

* [Continuous delivery](https://www.educative.io/module/lesson/introduction-to-microservices/gk4GMwnDV8G#Continuous-delivery)
  + [Phases](https://www.educative.io/module/lesson/introduction-to-microservices/gk4GMwnDV8G#Phases)
  + [Microservices facilitate continuous delivery](https://www.educative.io/module/lesson/introduction-to-microservices/gk4GMwnDV8G#Microservices-facilitate-continuous-delivery)
  + [Deployment must be automated](https://www.educative.io/module/lesson/introduction-to-microservices/gk4GMwnDV8G#Deployment-must-be-automated)

## Continuous delivery [#](https://www.educative.io/module/lesson/introduction-to-microservices/gk4GMwnDV8G#Continuous-delivery)

[Continuous delivery](http://continuous-delivery-book.com/) is an approach where software is continuously brought into production with the help of a continuous delivery pipeline. The pipeline brings the software into production via different phases. Have a look at the following drawing:

Let’s discuss each phase shown above.

### Phases [**#**](https://www.educative.io/module/lesson/introduction-to-microservices/gk4GMwnDV8G#Phases)

* Typically, the **software compilation**, **unit tests**, and **static code analysis** are performed in the **commit phase**.
* In the **acceptance test** phase, **automated tests** assure the correctness of the software regarding domain logic.
* **Capacity tests** check the performance at the expected load.
* **Explorative** tests serve to perform not-yet-considered tests or to examine new functionalities. In this manner, explorative tests can analyze aspects that are not yet covered by automated tests.
* In the end, the software is brought into **production**.

Microservices represent independently deployable modules. Therefore each microservice has its own continuous delivery pipeline.

This facilitates continuous delivery.

### Microservices facilitate continuous delivery [**#**](https://www.educative.io/module/lesson/introduction-to-microservices/gk4GMwnDV8G#Microservices-facilitate-continuous-delivery)

* The continuous delivery pipeline is **significantly faster** because the deployment units are smaller. Consequently, deployment is faster.
* Continuous delivery pipelines contain many test stages. The software has to be deployed in each stage. Faster deployments speed up the tests and therefore the pipeline.
* The **tests are also faster** because they need to cover fewer functionalities. Only the features in the individual microservice have to be tested, whereas in the case of a deployment monolith, the entire functionality has to be tested due to possible regressions.
* **Building up a continuous delivery pipeline is easier** for microservices. Setting up an environment for a deployment monolith is complicated. Most of the time, powerful servers are required. In addition, third-party systems are frequently necessary for tests. A microservice requires less powerful hardware. Besides, not many third-party systems are needed in the test environments.
  + However, running all microservices together in one integration test can cancel out this advantage. An environment suitable for running all microservices would require powerful hardware as well as integration with all third-party systems.
* The **deployment of a microservice poses a smaller risk** than the deployment of a deployment monolith. In the case of a deployment monolith, the entire system is deployed anew, and in the case of a microservice, only one module. This causes fewer problems since less of the functionality is being changed.

In summary, **microservices facilitate continuous delivery**. Even their support of continuous delivery can be reason enough to migrate a deployment monolith to microservices.

### Deployment must be automated [**#**](https://www.educative.io/module/lesson/introduction-to-microservices/gk4GMwnDV8G#Deployment-must-be-automated)

However, note that microservice architectures can only work when the deployment is automated! Microservices substantially increase the number of deployable units compared to a deployment monolith. This is only feasible when the deployment processes are automated.

Independent deployment means that the continuous delivery pipelines have to be completely independent. Integration tests conflict with this independence. They introduce dependencies between the continuous delivery pipelines of the individual microservices. Therefore, **integration tests must be reduced** to the minimum. Depending on the type of communication, there are different approaches to achieve this for synchronous and asynchronous communication.

In what phase of continuous delivery would the performance of an application be checked against the expected load?

###### C)Capacity test

Setting up an environment to integration test a microservices architecture can be complicated

###### True (An environment suitable for running all microservices would require powerful hardware as well as an integration with all third-party systems.)

In the next lesson, we’ll continue our discussion of the advantages of microservices.

# More on Advantages

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

**We'll cover the following**

* [Robustness](https://www.educative.io/module/lesson/introduction-to-microservices/mErkv5XwRn0#Robustness)
* [Independent scaling](https://www.educative.io/module/lesson/introduction-to-microservices/mErkv5XwRn0#Independent-scaling)
* [Free technology choice](https://www.educative.io/module/lesson/introduction-to-microservices/mErkv5XwRn0#Free-technology-choice)
* [Security](https://www.educative.io/module/lesson/introduction-to-microservices/mErkv5XwRn0#Security)
* [In general: isolation](https://www.educative.io/module/lesson/introduction-to-microservices/mErkv5XwRn0#In-general:-isolation)

## Robustness [#](https://www.educative.io/module/lesson/introduction-to-microservices/mErkv5XwRn0#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 [#](https://www.educative.io/module/lesson/introduction-to-microservices/mErkv5XwRn0#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 [#](https://www.educative.io/module/lesson/introduction-to-microservices/mErkv5XwRn0#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.

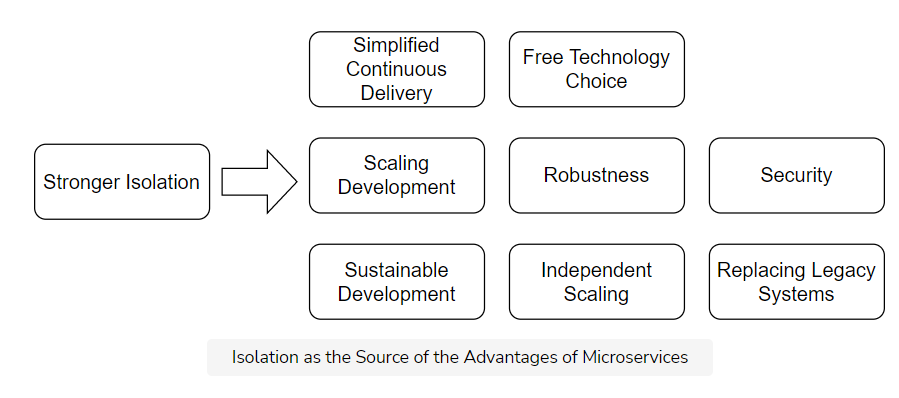
## Security [#](https://www.educative.io/module/lesson/introduction-to-microservices/mErkv5XwRn0#Security)

Microservices can be isolated from each other.

* 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 [#](https://www.educative.io/module/lesson/introduction-to-microservices/mErkv5XwRn0#In-general:-isolation)

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



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

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

###### B)The other microservices will keep running.

Free technology choice is a consequence of stronger isolation.

###### A)True

Suppose you are designing a small social network. Which microservices would be the most appropriate to divide it into from the given options?

###### B)

* Registration
* Searching for friends
* Adding friends
* Writing posts
* Reading posts

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.

# Tradeoffs, Prioritizing Advantages, & Levels

In this lesson, we'll look at some ways we can prioritize advantages of the microservice architecture and some potential trade-offs that should be considered.

**We'll cover the following**

* [Prioritizing advantages](https://www.educative.io/module/lesson/introduction-to-microservices/xVmDqqmPQPz#Prioritizing-advantages)
* [Microservices involve trade-offs](https://www.educative.io/module/lesson/introduction-to-microservices/xVmDqqmPQPz#Microservices-involve-trade-offs)
* [Two levels of microservices: Domain and technical](https://www.educative.io/module/lesson/introduction-to-microservices/xVmDqqmPQPz#Two-levels-of-microservices:-Domain-and-technical)
* [Typical numbers of microservices in a system](https://www.educative.io/module/lesson/introduction-to-microservices/xVmDqqmPQPz#Typical-numbers-of-microservices-in-a-system)

## Prioritizing advantages [#](https://www.educative.io/module/lesson/introduction-to-microservices/xVmDqqmPQPz#Prioritizing-advantages)

Which of the discussed reasons for switching to microservices is the most important **depends on the individual scenario**. The use of microservices in a greenfield system is the one exception.

More often, a deployment monolith is replaced by a microservice system (see [chapter 4](https://www.educative.io/collection/page/10370001/6518081205567488/4532691821133824)). In that case, different advantages are relevant.

* The easier **scaling of development** can be an important reason for the introduction of microservices in such a scenario. Often, it is impossible to work quickly enough with a large number of developers on a single deployment monolith.
* The **easy migration** away from the legacy deployment monolith facilitates the introduction of microservices in such a scenario.
* **Continuous delivery** is often an additional goal. The aim is to increase the speed and reliability with which changes can be brought into production.

The scaling of development is not the only scenario for a migration. For example, when a **single Scrum team** wants to implement a system with microservices, **scaling development would not be a sensible reason** since the organization of development is not large enough for this. However, **other reasons are possible**. Continuous delivery, technical reasons like robustness, independent scaling, free technology choice, or sustainable development all play a role in such a scenario.

In the end, it is important to **focus on increasing the business value**. Depending on the scenario, an advantage in one of the previously mentioned areas might make the company more profitable or competitive, for example, faster time to market or better reliability of the system.

## Microservices involve trade-offs [#](https://www.educative.io/module/lesson/introduction-to-microservices/xVmDqqmPQPz#Microservices-involve-trade-offs)

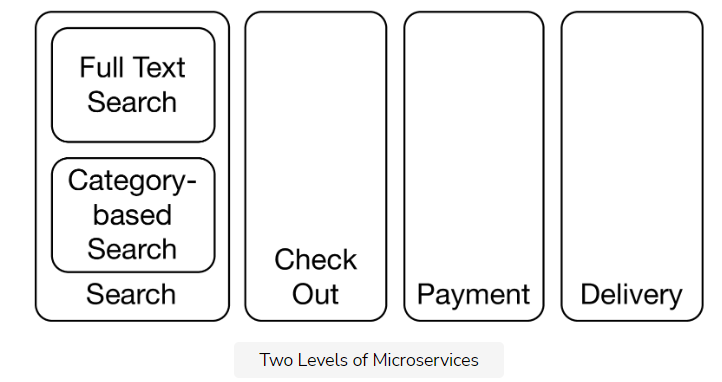
Depending on the aims, a team can compromise when implementing microservices.

* For example, when **robustness is the goal** of introducing microservices, the microservices have to be implemented as separate Docker containers.
  + Each Docker container can crash without affecting the other ones.
* **If robustness does not matter**, other alternatives can be considered. For example, multiple microservices can run together as Java web applications in one Java application server. In this case, they all run in one process and therefore are not isolated in respect to robustness. Still they are independently deployable.
  + A memory leak in any of the microservices will cause them all to fail.
  + However, such a solution is easier to operate and therefore might be the better trade-off in the end.

## Two levels of microservices: Domain and technical [#](https://www.educative.io/module/lesson/introduction-to-microservices/xVmDqqmPQPz#Two-levels-of-microservices:-Domain-and-technical)

The technical and organizational advantages point to **two levels** at which a system can be divided into microservices.

* A **coarse-grained division by domain** enables the teams to develop largely independently and allows them to roll out a new feature with the deployment of a single microservice if it concerns just this one domain, as it usually does. However, sometimes multiple domains are involved and more than one microservice must be deployed.
  + For example, in an e-commerce system, customer registration and the order process can be examples of such coarse-grained microservices.
* For **technical reasons** some microservices can be further divided. These microservices can then be scaled independently of the other microservices.
  + When, for example, the last step of the order process is under especially high load, this last step can be implemented in a separate microservice. The microservice belongs to the domain of the order process, but for technical reasons, it is implemented as a separate microservice. This is an example of a **technical division**.



The drawing above shows an example for the two levels. **Based on the domains**, an e-commerce application is **divided into** the microservices:

* Search
  + Full-text search
  + Category-based search
* Check out
* Payment
* Delivery

**Search is further subdivided**. The full-text search is separated from the category-based search.

* **Independent scaling** can be one reason for this. This architecture allows the system to scale the full-text search independently of the category-based search which is advantageous when both have to deal with different levels of load.
* Another reason could be the **use of different technologies**. The full-text search can be implemented with a full-text search engine, which is unsuitable for a category-based search.

**Typical numbers of microservices in a system**[#](https://www.educative.io/module/lesson/introduction-to-microservices/xVmDqqmPQPz#Typical-numbers-of-microservices-in-a-system)

It is difficult to state a typical number of microservices per system. Based on the divisions discussed in this chapter, 10-20 coarse-grained domains are usually defined, and each of these might be subdivided into one to three microservices. However, there are also systems with far more microservices.

Docker containers can help to make microservices independently deployable.

###### A)

True

Suppose you are designing a small social network and you come up with a domain-based microservice to **search posts**. Which of the following are plausible technical divisions of this?

###### B)

Search photo album posts / search text posts

In the next lesson, we’ll look at some challenges that the microservice architecture poses.

# Challenges

In this lesson, we'll look at possible challenges involved in a microservice architecture.

**We'll cover the following**

* [Increased operations effort](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Increased-operations-effort)
* [Must Be Independently Deployable](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Must-Be-Independently-Deployable)
* [Testing must be independent](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Testing-must-be-independent)
* [Difficult to change multiple microservices](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Difficult-to-change-multiple-microservices)
* [Lost overview](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Lost-overview)
* [Increased latency and failures](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Increased-latency-and-failures)
* [Weighing benefits and disadvantages](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Weighing-benefits-and-disadvantages)
* [Experiments](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Experiments)

## Increased operations effort [#](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Increased-operations-effort)

* The operation of a microservice system **requires more effort than running a deployment monolith**.
  + This is due to the fact that in a microservice system, many more deployable units exist that all have to be deployed and monitored.
  + This is feasible only when the operation is largely automated and the correct functioning of the microservices is guaranteed via appropriate monitoring.

## Must Be Independently Deployable [#](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Must-Be-Independently-Deployable)

* Microservices have to be **independently deployable**. For example, dividing them, into Docker containers is a prerequisite for this, but it is not enough on its own.
* Changes to interfaces must be implemented in such a way that an independent deployment of individual microservices is still possible.
  + For example, the microservice which implements the interface has to offer the new and the old interface. Then this microservice can be deployed without requiring that the calling microservice be deployed at the same time.

## Testing must be independent [#](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Testing-must-be-independent)

* Also, **testing must be independent**. When all microservices have to be tested together, one microservice can block the test stage and prevent the deployment of the other microservices making testing much harder.
* Due to the split into microservices, there are more interfaces to test, and **testing has to be independent for both sides of the interface**.

## Difficult to change multiple microservices [#](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Difficult-to-change-multiple-microservices)

* Changes that affect **multiple microservices** are **more difficult to implement** than the changes that concern several modules of a deployment monolith.
  + In a microservice system, such changes require several deployments. These deployments must be coordinated.
  + In the case of a deployment monolith, only one deployment would be necessary.

## Lost overview [#](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Lost-overview)

In a microservice system, the overview of the microservices can get lost. However, experience teaches that in practice, a sound domain-based division can restrict changes to one or a few microservices. Therefore, the overview of the system is less important because the interaction between the microservices hardly influences development due to the high degree of independence.

## Increased latency and failures [#](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Increased-latency-and-failures)

Microservices communicate through the network. Compared to local communication,

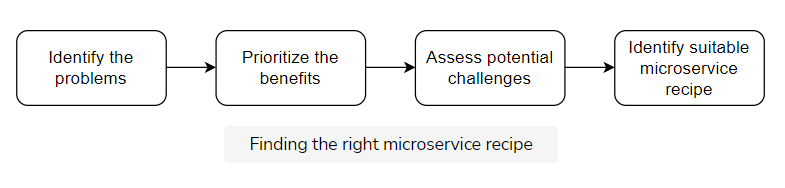
* The **latency is much higher**.
* It is also more likely that **communication will fail**.

A microservices system cannot rely on the availability of other microservices. This makes the systems more complex.

## Weighing benefits and disadvantages [#](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Weighing-benefits-and-disadvantages)

The most important rule is that microservices should only be used if they represent the simplest solution in a certain scenario.

The previously mentioned benefits should outweigh disadvantages resulting from the higher level of complexity for deployment and operation. Choosing a more complex solution is rarely a good idea.



## Experiments [#](https://www.educative.io/module/lesson/introduction-to-microservices/m7jjy5KjG4O#Experiments)

The following approach helps to find the right recipe to divide a system into microservices.

1. **Identify the problems** in your current system (for example, resilience, development agility, too slow deployment, and so on).
2. For the projects that you’ve worked on, **prioritize the benefits** of using microservices.
3. Weigh **which of the challenges** in this project could **pose a risk**.
4. Look at the possible technical and architectural solutions in the following chapters to determine the most sensible solutions for their requirements.

For the concrete division into microservices and for technical decisions, additional concepts are necessary. So, let’s discuss the question of how best to divide a system into microservices in the next chapter.

Suppose you’re designing an application where one microservice gets data from another. If this data fetching fails, the functionality of the app will be compromised. What would be the best course of action in this situation?

###### B)

Merge the two microservices into one. (Microservices communicate over the network, which is unreliable. The app should not depend on this communication.)

Automated deployment can save considerable effort with microservices.

###### A)

True

Microservices should only be tested independently. Testing them together should be avoided.

###### A)

True (Otherwise the deployment is not independent but microservices should be independently deployable.)

In the next lesson, we’ll look at a quick summary of what we have learned in this chapter.