### Microservices: advantages

* Microservice architecture gives developers the freedom to independently develop and deploy services
* A microservice can be developed by a fairly small team
* Code for different services can be written in different languages (though many practitioners discourage it)
* Easy integration and automatic deployment (using open-source continuous integration tools such as Jenkins, Hudson, etc.)
* Easy to understand and modify for developers, thus can help a new team member become productive quickly
* The developers can make use of the latest technologies
* The code is organized around business capabilities
* Starts the web container more quickly, so the deployment is also faster
* When change is required in a certain part of the application, only the related service can be modified and redeployed—no need to modify and redeploy the entire application
* Better fault isolation: if one microservice fails, the other will continue to work (although one problematic area of a monolith application can jeopardize the entire system)
* Easy to scale and integrate with third-party services
* No long-term commitment to technology stack
* Scale up on Services rather entire suite of monolithic applications.
* Improves fault isolation: Larger applications can remain largely unaffected by the failure of a single module.
* Eliminates long-term commitment to a single technology stack: If you want to try out a new technology stack on an individual service, go right ahead. Dependency concerns will be far lighter than with monolithic designs, and rolling back changes much easier. The less code in play, the more flexible you remain.
* Makes it easier for a new developer to understand the functionality of a service.

**Microservices: drawbacks**

Just because something is all the rage around the industry, doesn’t mean it has no drawbacks. Here’s a list of some potential pain areas associated with microservices designs:

* Developing distributed systems can be complex. By which I mean, because everything is now an independent service, you have to carefully handle requests travelling between your modules. There can be a scenario where one of the services may not be responding, forcing you to write extra code specifically to avoid disruption. Things can get more complicated when remote calls experience latency.
* Multiple databases, Multiple resources (JMS, DB, File) and transaction management can be painful.
* Testing a microservices-based application can be cumbersome. Using the monolithic approach, we would just need to launch our WAR on an application server and ensure its connectivity with the underlying database. But now, each dependent service needs to be confirmed before you can start testing.
* Deploying microservices can be complex. They may need coordination among multiple services, which may not be as straightforward as deploying a WAR in a container.
* Due to distributed deployment, testing can become complicated and tedious
* Increasing number of services can result in information barriers
* The architecture brings additional complexity as the developers have to mitigate fault tolerance, network latency, and deal with a variety of message formats as well as load balancing
* Being a distributed system, it can result in duplication of effort
* When number of services increases, integration and managing whole products can become complicated
* In addition to several complexities of monolithic architecture, the developers have to deal with the additional complexity of a distributed system
* Developers have to put additional effort into implementing the mechanism of communication between the services
* Handling use cases that span more than one service without using distributed transactions is not only tough but also requires communication and cooperation between different teams
* The architecture usually results in increased memory consumption
* Partitioning the application into microservices is very much an art

Of course, with the right kind of automation and tools, all the above drawbacks can be addressed.