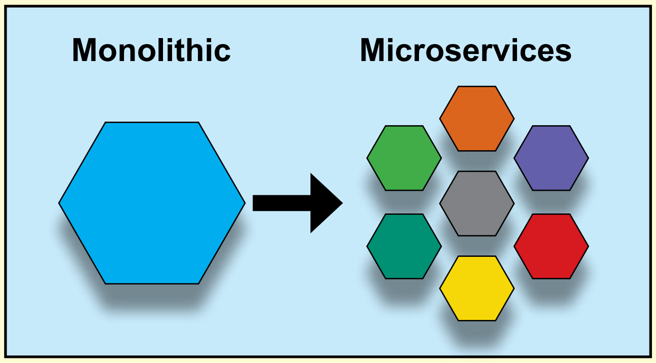
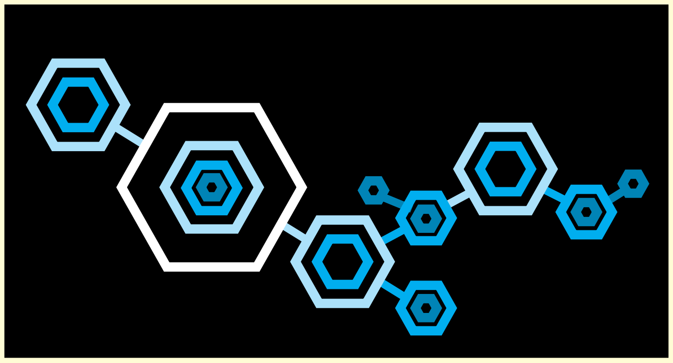
Services and Microservices

The development of a software system that supports one or more business processes can be an onerous undertaking for at least two reasons. First, it is likely to be complex because even a single business process may consist of many functions with multiple dependencies that have to be coordinated. Second, since both the software technology and the business process itself are likely to change over time, the design of the software should facilitate rather than encumber future changes. When faced with a problem that is both complex and subject to future changes, the time-proven approach is to divide the problem into less complex sub-problems. Not only are the sub-problems easier to solve, but they also form modules that can be replaced at a later date. If particular care is taken in the design of the modules so that they adhere to a standard protocol for input/output and that the tasks performed by any one module do not involve any other module, then we have created a very flexible and manageable solution approach.





The notion of a service and the entire concept of service-oriented architecture in computer software development is based on this approach. The difference between a service and a microservice is essentially one of scope. The more focused the tasks performed by the module the more likely that the module is completely self-sufficient and therefore can perform its tasks independently of any other module. This means that the module can be removed without directly impacting the ability of any other module to execute its tasks, except that the services provided by that module may no longer be available in the system. More importantly, the module can be replaced by another module as long as the new module adheres to the same input/output protocol.

In more technical terms a service or microservice has a standard service contract interface that is decoupled from its implementation. Ideally, the standard service contract should make no assumptions about the programming language and platform of the consumers of the results produced by the service. This allows software programs developed with different technologies to consume the same service. Besides providing technology independence which is critical for supporting software interoperability in a technologically heterogeneous environment, the decoupling creates two other conditions. First, the methods used by the service to create its results are of no consequence to any other component of the software system. Therefore, a service can be replaced with another service that may use a completely different method for generating the expected results. Second, the request sent to a service must be self-contained because the service has no knowledge of how the results that it produces are going to be used by either the consumer of its services or the system at large. In technical terms this is referred to as a stateless service. A stateless service has no need to retain any memory of a request after it has fulfilled it, since each new call for services is independent from all previous calls.