Microservices Architecture

You’re designing a server-side, multi-user application. You want your application to be modular, and you want the modules to be independent. Your application modules need to be capable of composition, scalability, and continuous deployment.

How do you architect an application as a set of independent modules?

Several factors make architecture of independent modules challenging:

- Modular code doesn’t tend to stay modular when it’s being developed by one big team and runs in one monolithic process.
- You don’t want to have to wait until development of all modules is complete in a large system before releasing some of the functionality if it is independently useful to the business.
- When a single business function is divided into separate modules, those modules must evolve in lockstep.
- Separate business functions can evolve at different rates.
- For a component to be modular, not only must its code be modular, but its data must be isolated and logically cohesive as well.

Therefore,

Design the application with a Microservices Architecture, which is a set of modules where each is an independent business function with its own data, developed by a separate team and deployed in a separate process. For instance, a large retailing website may have separate microservices for different functions, such as catalog item search, cart checkout, and customer service. These can be developed and released on different timelines.

By developing a large application as a set of independent microservices, the individual services can be released and likewise evolve at their own pace. You don’t have to wait for the entire system to be complete before the business can begin to realize value from the system. There are different styles of microservices for different needs, for instance adapting to existing systems vs. implementing new business logic, but all share the same traits of independent scaling and independent development. This allows you to, for instance, take advantage of Polyglot Development to give teams more autonomy in choosing the right tools for each specific job.

In fact, the microservices approach gives you better control over scalability than a traditional approach can give you. By only scaling each microservice to the level at which they are required in order to handle the requests made for that specific service, you will be able to more efficiently use your computing resources than in the traditional model of scaling the entire monolith to the largest number of instances needed to handle requests for the most commonly used functions.

Splitting up business functions into multiple small Domain Microservices, allows each microservice to be developed independently by a small team. Where you need to adapt to an existing interface, either external or internal, you can write an Adapter Microservice. Backends for Frontends (BFFs) are used to mediate between different client types and Business Microservices.

Microservices can be deployed independently with zero downtime for continuous deployment. Microservices can scale and fail independently. Microservices developed and deployed independently will tend to become and remain separate modules. However, often a monolith is easier to monitor and manage than multiple microservices, so you’d better get good at Cloud Native DevOps.