Entering this class with an extremely limited understanding of microservices, this article exposed me to its architecture and the advantages and disadvantages that come with adopting it. The microservice architecture, also known as MSA, offers numerous benefits, such as agility (the ability to flexibly adapt to changes- i.e. updates or new features), scalability (the ability to efficiently manage an increasing workload), and distributed development. However, despite these advantages, the authors caution the readers that adopting this architecture requires careful consideration of consequences in cultural aspects. Examples of these consequences include needing to manage unsettling technical and cultural changes, which requires preparing strong application architecture skills and having the readiness to adopt new designs in an Agile and DevOps setting, ensuring that each microservice investment has a specific business outcome, and giving the team self-autonomy in order to allow them to build the components that they were assigned. To adopt MSA successfully, significant investments in people, processes, and platforms are therefore essential, which all come with their own difficulties and costs.

I also learned that MSA, which was popularized mainly by Netflix, Amazon, and Twitter, facilitates continuous delivery practices and enables organizations to deploy new features quickly, which is incredibly important when it comes to competition. The granularity of microservices allows for independent development (which means that each component can be worked on and updated separately), reducing risks associated with each deployment. However, it is also important to note that although this is helpful, it can also introduce complexities and challenges, such as managing dependencies between microservices.

Cohesion is another crucial principle of MSA mentioned in the article, aiming to reduce dependencies between microservices. However, publishing a service’s API for reuse creates dependencies that can impact the ability to make changes. Microservices thus, implement individual features and never publish their API for external consumption. Miniservices are similar to microservices with the only difference being that they may implement more than one feature, while macroservices incorporate a specific function into a monolithic (large, unified software application) and expose it via a published API.

The authors conclude the article, emphasizing yet again the importance of approaching MSA adoption cautiously and only implementing it if specific requirements are met. A popular alternative is miniservices, offering a low-code application and cost-effective solution. The main message emphasizes not succumbing to hype and adopting MSA only when extreme agility and scalability requirements are present. Organizations are advised to start with basic SOA adoption, encapsulate functionality as macroservices, refactor into miniservices, and consider microservices only where necessary. I also never knew that mixing the different paradigms- microservices, monolithic, and miniservices architectures- is encouraged for a pragmatic approach to development, which is surprising because I didn’t think that it was possible to mix the three.