Decoupling
1
Componentization
2
Business Capabilities
3
Autonomy
4
5
Continous Delivery
6
Responsibility
7
Decentralized Governance
8
Agility

1. **Decoupling** – Services within a system are largely decoupled. So the application as a whole can be easily built, altered, and scaled
2. **Componentization** – Microservices are treated as independent components that can be easily replaced and upgraded
3. **Business Capabilities** – Microservices are very simple and focus on a single capability
4. **Autonomy** – Developers and teams can work independently of each other, thus increasing speed
5. **Continous Delivery** – Allows frequent releases of software, through systematic automation of software creation, testing, and approval
6. **Responsibility** – Microservices do not focus on applications as projects. Instead, they treat applications as products for which they are responsible
7. **Decentralized Governance** – The focus is on using the right tool for the right job. That means there is no standardized pattern or any technology pattern. Developers have the freedom to choose the best useful tools to solve their problems
8. **Agility** – Microservices support agile development. Any new feature can be quickly developed and discarded again

Advantages Of Microservices
1. **Independent Development** – All microservices can be easily developed based on their individual functionality
2. **Independent Deployment** – Based on their services, they can be individually deployed in any application
3. **Fault Isolation** – Even if one service of the application does not work, the system still continues to function
4. **Mixed Technology Stack** – Different languages and technologies can be used to build different services of the same application
5. **Granular Scaling** – Individual components can scale as per need, there is no need to scale all components together

Conway's law
"organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations."

The law is based on the reasoning that in order for a software module to function, multiple authors must communicate frequently with each other. Therefore, the software interface structure of a system will reflect the social boundaries of the organization(s) that produced it, across which communication is more difficult. Conway's law was intended as a valid sociological observation, although sometimes it's used in a humorous context. It was dubbed Conway's law by participants at the 1968 National Symposium on Modular Programming.[3]