**Principles of microservice:**

* Decentralization: Services are developed, deployed, and managed independently.
* The disadvantage of monolith is in the same application, whenever need to deployed single service it needs to restart the whole application. The microservice architecture resolves this problem by separating the service. If we need to update, or deploy it just deploy in single service. Besides that, if it failed in one service not whole application, improving system resilience.
* Componentization: Services are treated as independent components that can be easily replaced and upgraded.
* Service will be implemented, deployed, and scaled independently, not affect to other service. Also it can be replaced and upgrade individual without affect the entire system. Because it is as single component so easy to debugging and testing on it, do component test by mock api.
* Autonomy: Teams work independently on each service, reducing the coordination overhead.
* Each team handles it own service, full decision to make in service, no need to wait other team, it makes the implementation and deployment time shortly. It can be done in a sprint.
* Technology Diversity: Teams can choose the best tool for their specific needs, fostering innovation.
* Based on the specific request, allow team to chose the best suite approach for each service, lead to optimized solution. Provides in flexibility the technical choice, reduce dependency the specific vendor and platform.

**Advantages of microservice:**

* Scalability: Services can be scaled independently, allowing for more efficient use of resources.
* Cause services is independent so we can allocate the resource(cpu, ram, database) based on the specific of that service, it leads to optimized resource utilization.
* Cost reduction: reduce waste cost because only scalable the needed services.
* Improve performance: Increase performance of application by scale the necessary services.
* Resilience: Faults in one service do not impact others, improving overall system robustness.
* Improve system stability: enhance the system ability to recover from failure and continue functioning without affect to other services.
* Isolate the faiure: make user failure only in one service and not propagate to other services make system availability and reliability.
* Technological Agility: Allows the adoption of new technologies and processes without overhauling the entire system.
* Teams can choose the best tool for their specific needs, fostering innovation and enabling the adoption of the latest technologies
* Because of modular and decoupled nature of microservices, it is easier to maintain and update individual services without affecting the entire system

**Challenges of microservice:**

* Complexity: Increased operational and management complexity.
* Monitoring and Logging: Monitoring and logging across multiple services can become complex, requiring centralized logging and monitoring solutions
* Service Orchestration: In a microservices architecture, multiple services often need to collaborate to complete a single operation, leading to complex service orchestration and coordination
* Data Integrity: Ensuring data consistency across services can be challenging.
* Distributed Data Management: Managing data across multiple services can lead to inconsistencies and data integrity issues.
* Network Issues: Dependency on network latency and load balancing.
* Network Latency: Network latency can impact the performance and responsiveness of microservices.
* Load Balancing: Efficient load balancing mechanisms are required to distribute incoming requests evenly across multiple instances of a service.
* Service Discovery and Communication: Dynamic service discovery and efficient communication between services are essential for the seamless operation of microservices.
* Skill Set: Requires a broad set of skills from development teams, including DevOps capabilities.
* DevOps Capabilities: Operations teams need to have strong DevOps capabilities to manage and orchestrate multiple services, implement CI/CD pipelines, and manage infrastructure as code.
* Monitoring and Troubleshooting: Operations teams need to have advanced monitoring and troubleshooting skills to manage and monitor microservices and resolve issues efficiently.