**Introduction to Microservice**

**Principles of Microservices**

**Decentralization:** each service can implement, build, and deploy independently; we can manage source code/database separately. If one of service down, it is not impacted to other services.

**Componentization:** Cause service is independent with others, so we can be easy to upgrade the code, change or replace any components of that service.

**Autonomy:** Each microservice is typically owned and maintained by a small, cross-functional team, which has the autonomy to make decisions about the technology stack, architecture, and deployment process for their service. This reduces coordination overhead and allows teams to move quickly and iterate on their services without being bottlenecked by other teams.

**Technology Diversity:** inside each service, each team can choose a specific programming language for their strengthen working like service A using Java, service B using C#, service C using PHP or any frameworks, and libraries without affecting the rest of the system. Moreover, the database storage can be diversity like PostgreSQL, MYSQL, MongoDB for each service.

### **Advantages of Microservices**

### **Scalability:** Services can be scale independently, it is not belonged to other services, and it can use efficient resources.

**Resilience:** Faults, issue, bug, or problem in a service can be isolate and is not impact to other services; improving overall system robustness.

**Technological Agility:** Allows the adoption of new technologies and processes without overhauling the entire system.

### **Challenges of Microservices**

**Complexity:** For monolithic we just need a service and it’s containing all the business purposes/logic, it is simply to implement and deployment. But for the microservices we need to increase the operation, separate service to multiple services based on each purpose, build the connection, setup the configuration, setup the service discovery… With numerous services to develop, deploy, and maintain, the overall system architecture becomes more intricate. Managing the interactions between microservices, ensuring proper communication, and orchestrating deployments across various environments can become challenging tasks.

**Data Integrity:** maintaining data consistency across services can be significant challenges. Since each services have its own database or storage, ensuring database integrity and consistency data need a lot of implement and flow.

**Network Issues:** Microservices rely heavily on network communication for inter-service communication. This dependency on the network introduces challenges related to network latency, reliability, and load balancing. Service degradation or failures can occur due to network issues, impacting the overall performance and reliability of the system. Implementing robust networking solutions, such as service discovery, circuit breakers, and retries, is essential to mitigate these challenges.

**Skill Set:** Adopting microservices requires a diverse skill set from development teams. Developers need to have expertise not only in software development but also in areas such as containerization, orchestration, continuous integration/continuous deployment (CI/CD), monitoring, and infrastructure management. Additionally, teams must possess DevOps capabilities to effectively manage the entire lifecycle of microservices, from development to deployment and operation. Acquiring and maintaining these skills can be a significant challenge for organizations transitioning to a microservices architecture. Training programs, knowledge sharing, and collaboration across teams can help address this challenge.