# Introduction to Microservices

## Introduction

* Define microservices architecture: An architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities.
* Briefly compare with monolithic architecture to highlight the context and evolution

## Principles of Microservices

* Decentralization: Services are developed, deployed, and managed independently.

+ This allows each team to work autonomously on different services without being tightly coupled to each other. Each service is responsible for a specific business capability and can be develop, test and deploy independently. This will enable faster release cycles, because one service does not require the other service.

* Componentization: Services are treated as independent components that can be easily replaced and upgraded.

+ Each service encapsulates a specific business or functionality and interactions between services are through Rest APIs or Messaging queue. By componentization services, project can replace or upgrade individual components without impacting the entire system, and this makes it easier to change to new technologies, scale and evolve parts of the system independently.

* Autonomy: Teams work independently on each service, reducing the coordination overhead.

+ Each team is responsible for a specific service or set of services and they have the freedom to make any decisions on the service about the technology used, the architecture and development, release process. This autonomy reduces coordination costs across teams and help innovation faster without being constrained by dependencies on other teams.

* Technology Diversity: Teams can choose the best tool for their specific needs, fostering innovation.

+ By allowing teams to choose the best tools and technologies for their specific needs, each service can be build using different programming languages, frameworks, databases and development process. This fosters innovation and allows to leverage the strengths of different technologies to solve specific problems more easily.

## Advantages of Microservices

* Scalability: Services can be scaled independently, allowing for more efficient use of resources.

+ Services can be scaled independently, that means that resources can be scaled and allocated more efficiently based on the demand of that services, for instance the services experiencing high traffic need to be scaled out and the others do not need. By scaling only the necessary services, project can optimize resource usage and reduce costs. The scaling horizontally provides better performance and reliability because it distributes the load across multiple instances of the service.

* Resilience: Faults in one service do not impact others, improving overall system robustness.

+ Microservices promote resilience by isolating faults within individual services. If one service fails or experiences issues, it does not necessarily impact the functionality of other services or the entire system. Since services are decoupled and communicate through Rest APIs, failures are contained within the affected service boundary. This isolation prevents cascading failures and improves the overall robustness of the system.

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

+ Microservices architecture enables organizations to adopt new technologies and processes more easily without overhauling the entire system. Each service can be developed, deployed, and updated independently, allowing teams to experiment with new tools, frameworks, or programming languages. Teams can leverage the latest technologies to address specific business needs or take advantage of emerging trends without being constrained by monolithic architectures.

## Challenges of Microservices

* Complexity: Increased operational and management complexity.

+ Microservices introduce increased operational and management complexity compared to monolithic architectures. With a distributed system composed of many individual services, there are more components to deploy, monitor, and manage.

+ Managing the interactions and dependencies between services, as well as ensuring their overall health and availability, can become challenging as the system grows in size and complexity.

+ organizations may need to invest in sophisticated tooling and infrastructure to effectively manage a microservices-based architecture, which can further contribute to complexity.

* Data Integrity: Ensuring data consistency across services can be challenging.

+ Ensuring data consistency and integrity across multiple microservices can be challenging, especially in distributed environments where each service may have its own database or data store.

+ Maintaining data consistency requires careful coordination and synchronization mechanisms, such as distributed transactions, event-driven architectures, or eventual consistency models.

+ Inconsistencies in data across services can lead to issues such as data corruption, stale data, or incorrect application behavior, posing significant challenges for developers and operations teams.

* Network Issues: Dependency on network latency and load balancing.

+ Microservices rely heavily on network communication for inter-service communication, which introduces dependencies on network latency, reliability, and load balancing.

+ Network issues such as latency spikes, packet loss, or network partitions can impact the performance and availability of microservices-based systems.

+ Service degradation or failures due to network issues can have cascading effects on dependent services, potentially leading to system-wide outages or performance degradation.

* Skill Set: Requires a broad set of skills from development teams, including DevOps capabilities.

+ Developing and operating microservices-based systems requires a broad set of skills and expertise from development teams, including proficiency in distributed systems, containerization, orchestration, and DevOps practices.

+ Teams need to be capable of designing, building, deploying, and monitoring individual services independently, while also ensuring seamless integration and coordination between services.

+ Organizations may need to invest in training and upskilling their teams to acquire the necessary skills and knowledge to effectively work with microservices-based architectures.

## Monolithic vs Microservices Architecture

|  |  |
| --- | --- |
| Monolithic Architecture | Microservices Architecture |
| Consists of a single codebase with multiple modules within according to the business functionalities | Consists of individual services with each service being responsible for exactly one functionality |
| Do not need expert domain knowledge for development | Risky to implement without domain expertise and container knowledge |
| Easier development | Relatively complex deployment |
| Updating the system is a tedious process which would need the entire system to be redeployed | Only the service which is updated needs to be redeployed |
| Reusing the modules from one software into other software systems is difficult | Microservices can be easily used in development of other software. |