.NET Core Microservices – True Ultimate Guide

Section 1 – Introduction to Microservices – Cheat Sheet

Introduction to Microservices

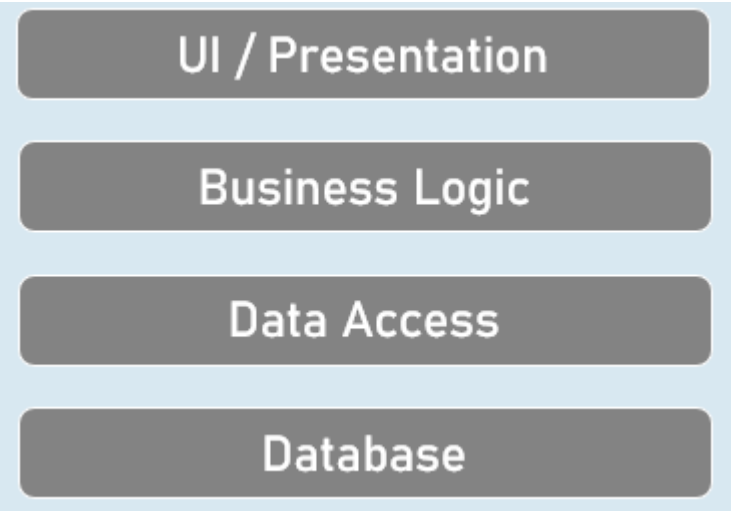
"Microservices" is a software architectural approach where applications are structured as a collection of smaller, single responsibility, loosely coupled, independently deployable services; each dedicated to performing a distinct function.

Each microservice operates with a clear boundary, owning or encapsulating both of its functionality and data store.

The logic & data of a microservice should be insulated and should NOT leak outside of it.

Monolithic Architecture

Monolithic architecture is an architectural approach in software development, in which the application is built as a single, tightly integrated unit, where all components and functions of the application such as UI, business logic and data access are tightly-coupled and run within a single process or application instance with a single data store.



Drawbacks of Monolithic Architecture

1. Source Code Management

Managing a large, tightly-integrated codebase can be challenging, leading to difficulties in collaboration, testing, and version control.

2. Team Management

As the size of development teams grows, managing larger development teams in a monolithic architecture can become complex, resulting in difficulties in version control, communication and project management.

3. Maintenance Issues

Making changes to a monolithic application can be complicated and risky. A small change in one part of the application might affect other parts, requiring extensive testing. This makes maintenance and updates slower and more error-prone.

4. Deployment Issues

Deploying updates can result in downtime during updates, complex releases, resource-intensive processes during deployment, and longer deployment cycles because of the need for comprehensive testing and fixing potential complications during updates.

5. Scalability Issues

Monolithic applications can be challenging to scale. When you need to handle increased load, you have to scale the entire application rather than just the specific parts that need it. This can be inefficient and costly.

6. Technology Stack

In a monolithic architecture, you are often limited to a single technology stack or programming language for the entire application, making it difficult to choose the best tool for each specific task.

Design Principles of Microservices

Agility is the underlying driving force of microservices, with 'independent and autonomous' as the central design principle.

1. Agility

The core idea of microservices is to make the application swiftly respond to evolving requirements & enable rapid development, deployment, and adaptation to changes in a dynamic and flexible manner.

2. Decomposition

The application is decomposed into smaller, manageable services; each focused on a specific business function.

Microservices are designed with Single Responsibility Principle (SRP).

3. Independence

Microservices are designed to be independently deployable.

This means that changes and updates to one microservice should not necessitate changes to other microservices.

Each microservice has its own database / data store.

4. Autonomy

Each service team has a high degree of autonomy.

They can choose the technology stack, database, and deployment strategies that best suit their service's requirements.

5. API-based Communication

Services communicate with each other through well-defined RESTful APIs, often over lightweight protocols like HTTP or gRPC.

6. Scalability

Microservices enable horizontal scalability.

You can scale individual services independently based on their specific performance and resource demands.

7. Resilience

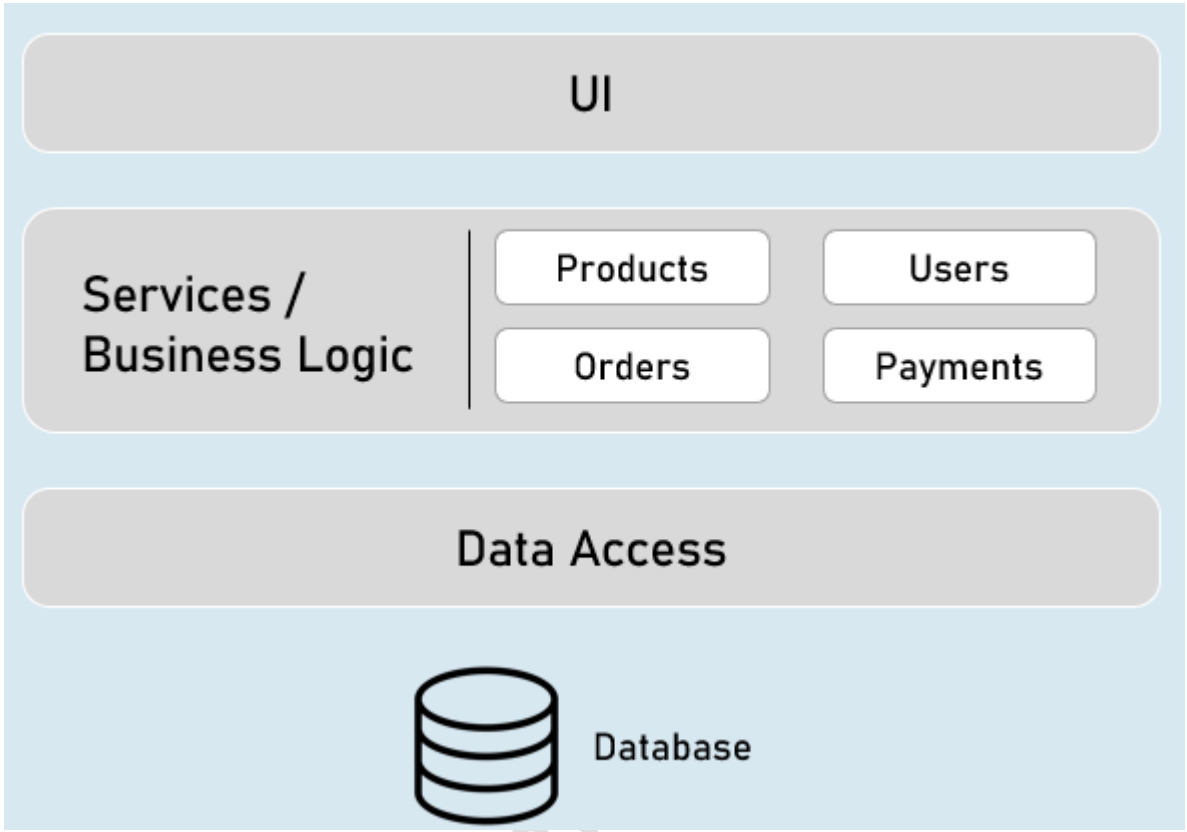
Resilience is about making the service to be able to handle failures gracefully, and the failure of one service should not bring down the entire application.

Techniques like circuit breakers and retries are often employed.

8. Testing

Each service can be tested individually, without affecting other services.

Breaking Monolithic Arch to Microservices



Microservices Best Practices

- Keep SRP, Independent, Isolated, Autonomy, Loosely coupled nature of Microservices in mind while designing Microservices.
- Enable logging, tracing and monitoring from the beginning of service implementation.
- Add resilience and fault tolerance to your services from the beginning of service implementation.
- Enable versioning and backward compatibility to your services from the beginning of service implementation.