**Q1**) if two or more services are closely related then on what basis you will decide that both should be clubbed into a single microservice or will keep and consider them as different microservices in microservices architecture.

**Ans**)

Deciding whether to club closely related services into a single microservice or keep them as separate microservices is a critical design decision in microservices architecture. This decision can be guided by several factors, including business domain boundaries, scalability, development and deployment independence, and fault tolerance. Let's explore these factors through an example.

Example Scenario: E-commerce Application

In an e-commerce application, let's consider two related services:

**Order Service**: Handles order creation, updates, and retrieval.

**Payment Service**: Manages payment processing for orders.

Factors to Consider

1. **Business Domain Boundaries (Bounded Contexts)**

**Order Service**: Focuses on managing the lifecycle of orders.

**Payment Service**: Deals with the financial transactions related to orders.

If the business logic for orders and payments is distinct and doesn't overlap significantly, it is a strong indication to keep them separate. The Order Service might deal with inventory checks, order history, and shipment details, while the Payment Service handles transaction validation, fraud detection, and payment gateway interactions.

2. **Scalability**

**Order Service**: Might experience high traffic during peak shopping times (e.g., Black Friday).

**Payment Service**: Might need to scale differently based on transaction volume and complexity.

Different scalability requirements suggest separation. The Payment Service may require more robust security and compliance measures and might need to scale horizontally to handle payment processing load independently from order management.

3. **Development and Deployment Independence**

**Order Service**: Requires frequent updates due to changing business rules or new features.

**Payment Service**: Requires updates for compliance with payment processors and security patches.

If the services are updated and deployed on different schedules, separating them can reduce the risk of impacting each other. This allows the Payment Service to undergo rigorous security testing and compliance checks without affecting the Order Service's deployment cycle.

4. **Fault Tolerance**

**Order Service**: Should be available for users to browse and place orders.

**Payment Service**: Must be highly reliable and secure, but its failure should not bring down the entire order management system.

Decoupling services improves fault tolerance. If the Payment Service fails, the Order Service can still function, allowing customers to place orders that can be processed later.

Decision

Based on the factors above, it is generally advisable to **keep the Order Service and Payment Service as separate microservices**. Here's why:

**Bounded Contexts**: They represent different business domains with distinct responsibilities.

**Scalability**: They have different scalability needs.

**Deployment Independence**: They may have different update frequencies and deployment schedules.

**Fault Tolerance**: Separating them improves the overall system resilience.