Microservice design patterns help ensure scalability, reliability, and maintainability of systems. Here are some commonly used patterns:

**1. API Gateway**

* Centralizes client requests and routes them to appropriate microservices.
* Handles authentication, logging, rate limiting, and response aggregation.

**6. Database Per Service**

* Each microservice maintains its own database to ensure autonomy.
* Avoids data sharing that could lead to tight coupling.

**7. CQRS (Command Query Responsibility Segregation)**

* Separates the read and write operations of a service into different models.
* Enhances scalability and performance.

**5. Saga Pattern**

* Coordinates distributed transactions across microservices using asynchronous communication.
* Ensures that workflows compensate for failures in long-running transactions.

**3. Circuit Breaker**

* Protects systems from cascading failures by stopping calls to failing services temporarily.
* Improves overall system resilience.

**2. Service Discovery**

* Manages dynamically assigned network locations of microservices.
* Ensures service instances can discover and communicate with each other efficiently.

**4. Event Sourcing**

* Captures all changes to the application state as a sequence of events.
* Allows reconstructing application state at any point and supports eventual consistency.

**8. Strangler Pattern**

* Gradually replaces legacy systems by building new functionality in microservices.
* Redirects traffic from the old system to the new microservices.

**9. Bulkhead**

* Isolates critical microservices from others to prevent failures from spreading.
* Improves resilience and reliability.

**10. Health Check**

* Continuously monitors the health of microservices to detect failures early.
* Often combined with automated recovery or scaling mechanisms.

These design patterns are essential tools in architecting microservice-based applications. Do you have a specific use case in mind you'd like to explore further?