12 Microservices Design Patterns to Know Before the System Design Interview

DesignGurus.io

1 Strangler Fig Pattern

Facilitates the gradual replacement of a monolithic system with microservices, ensuring a smooth and risk-free transition.

DesignGurus.io

2 API Gateway Pattern

Centralizes external access to your microservices, simplifying communication and providing a single entry point for client requests.

3 Backends for Frontends Pattern (BFF)

Creates dedicated backend services for each frontend, optimizing performance and user experience tailored to each platform.

DesignGurus.io

4 Service Discovery Pattern

Enables microservices to dynamically discover and communicate with each other, simplifying service orchestration and enhancing system scalability.

5 Circuit Breaker Pattern

Implements a fault-tolerant mechanism for microservices, preventing cascading failures by automatically detecting and isolating faulty services.

DesignGurus.io

6 Bulkhead Pattern

Isolates microservices into separate partitions, preventing failures in one partition from affecting the entire system and enhancing system resilience.

7 Retry Pattern

Enhances microservices' resilience by automatically retrying failed operations, increasing the chances of successful execution and minimizing transient issues.

8 Sidecar Pattern

Attaches additional components to your microservices, providing modular functionality without altering the core service itself.

9 Saga Pattern

Manages distributed transactions across multiple microservices, ensuring data consistency while maintaining the autonomy of your services.

10 Event-Driven Architecture Pattern

Leverages events to trigger actions in your services, promoting loose coupling between services and enabling real-time responsiveness.

DesignGurus.io

11 Command Query Responsibility Segregation Pattern

Separates the read and write operations in a microservice, improving performance, scalability, and maintainability.

12 Configuration Externalization Pattern

Provides a method to externalize the configuration from the code, enabling microservices to be reconfigured without the need for recompilation or redeployment.

Learn about the Microservices Design Patterns in "Grokking Microservices Design Patterns" from **DesignGurus.io**