HTWK Hochschule für Technik, Wirtschaft und Kultur Leipzig

IM Fakultät Informatik und Medien

Resilienz und Fehlertoleranz in verteilten Systemen

Modul „Software Engineering“

14. Januar 2025

Derhachov, Schmidt, Westholt

HTWK Leipzig, FIM

Gliederung

1. Resilienz und Fehlertoleranz
   - Begriffsklärung
   - Motivation
2. Strategien
   - Resilienzstrategien
   - Fehlertoleranzstrategien
3. Fazit
4. Diskussion
5. Quellen

Strategien

- Strategien zur Resilienzsteigerung in verteilten Systemen

Strategien

- Strategien zur Resilienzsteigerung in verteilten Systemen
- Fehlertoleranz und Ausfallsicherheit

Strategien

- Strategien zur Resilienzsteigerung in verteilten Systemen
- Fehlertoleranz und Ausfallsicherheit
- Praktische Implementierungen wie Circuit Breaker, Retry-Muster

Strategien

- Strategien zur Resilienzsteigerung in verteilten Systemen
- Fehlertoleranz und Ausfallsicherheit
- Praktische Implementierungen wie Circuit Breaker, Retry-Muster
- Ziel: Absicherung moderner Anwendungen gegen Störungen

Einleitung und Motivation

Motivation

- Schutz vor Ausfällen in geschäftskritischen Anwendungen
- Reduktion betrieblicher Kosten und Steigerung der Nutzerzufriedenheit
- Anforderungen an Verfügbarkeit in regulierten Branchen

Technologische Entwicklungen

- Verbreitung von Microservices und Cloud-Technologien
- Herausforderungen durch Kommunikationsausfälle und Spitzenlasten
- Bedarf an innovativen Resilienzmustern

Resilienzstrategien

- Redundanz
- Partitionierung
- Skalierung

Fehlertoleranzstrategien

- Fehlerbehandlung und -isolierung
- Nutzung von Fallback-Mechanismen
- Vermeidung kaskadierender Fehler

Circuit-Breaker

- Isoliert fehlerhaftxe Dienste
- Unterbricht Anfragen bei wiederholtem Fehler.
- Verhindert kaskadierende Ausfälle

Circuit-Breaker: Zustandsdiagramm

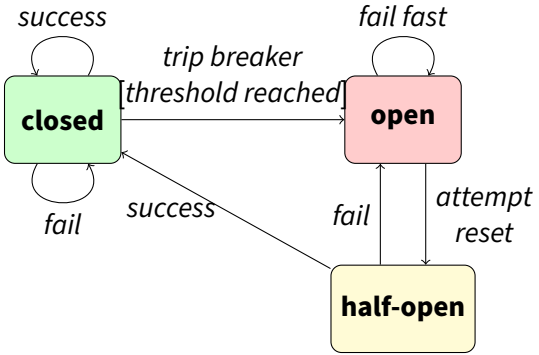

Abbildung 1: Circuit Breaker Zustandsdiagramm

Circuit-Breaker: Vorteile

- Verhindert kaskadierende Ausfälle in verteilten Systemen.
- Verbesserte Systemstabilität durch Isolierung fehlerhafter Dienste.
- Bessere Benutzererfahrung durch Fallback-Mechanismen.
- Unterstützt Resilienz und Wiederherstellung in kritischen Systemen.

Circuit-Breaker: Nachteile

- Erhöhte Komplexität in der Implementierung und Wartung.
- Risiko von Fehlkonfiguration (z. B. falsche Schwellenwerte).
- Zusätzlicher Overhead durch Überwachung und Statusverwaltung.
- Fallback-Daten können veraltet oder ungenau sein.

Retry-Muster

- Automatisches Wiederholen fehlgeschlagener Operationen
- Nutzung von Exponential Backoff
- Verbesserung der Resilienz bei temporären Fehlern

Retry-Muster: Sequenzdiagramm

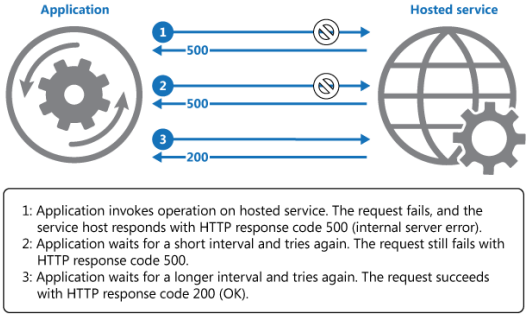

Abbildung 2: Sequenzdiagramm des Retry Patterns

Retry-Muster: Vorteile

- Reduziert die Wahrscheinlichkeit eines vollständigen Anwendungsabsturzes bei vorübergehenden Fehlern.
- Verbessert die Zuverlässigkeit, indem kurzfristige Probleme (z. B. Netzwerkprobleme) automatisch überwunden werden.
- Ermöglicht ein einheitliches Fehlerbehandlungsmodell in einer Anwendung.

Retry-Muster: Nachteile

- Erhöhte Komplexität in der Implementierung und Wartung.
- Verzögert die Gesamtverarbeitung, wenn ein Vorgang wiederholt fehlschlägt.
- Kann echte, dauerhafte Fehler verschleiern, wenn nur wiederholt wird, ohne die Ursache zu analysieren.
- Nicht jeder Fehler ist vorübergehend (z. B. Authentifizierungsfehler), was zu unnötigen Wiederholungen führt.

Kombination mit Circuit-Breaker

- Stoppt Wiederholungen bei permanenten Fehlern
- Ermöglicht Systemen, sich zu erholen
- Optimiert Ressourcennutzung

Load Balancing

- Verteilung der Last auf mehrere Server
- Strategien: Round Robin, Least Connection, Resource Based
- Verbesserung von Leistung und Ausfallsicherheit

Load Balancer: Architekturdiagramm

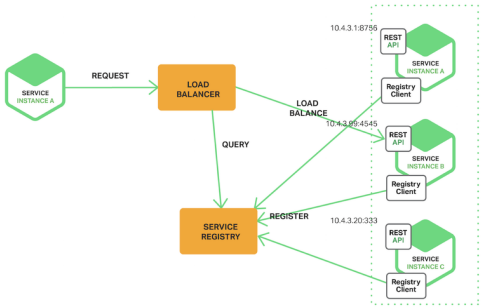

Abbildung 3: Architekturdiagramm mit einem zentalen Load Balancer

Health Checks bei Load Balancing

- Überwachung der Zustände von Diensten
- Vermeidung von Überlastungen
- Umgang mit fehlerhaften Knoten

Was wurde gemacht?

- Analyse und Implementierung von Resilienzstrategien
- Fallstudien und Praxisbeispiele
- Implementierungen in Python

Fallstudie: Netflix

- Nutzung von Hystrix für Circuit-Breaker
- Dynamisches Load Balancing
- Skalierung und Fehlertoleranz

Fazit

- Circuit Breaker und Retry-Muster als effektive Mechanismen
- Kombination verschiedener Strategien verbessert Resilienz
- Bedeutung von agilen Methoden zur kontinuierlichen Verbesserung

Diskussion

- Erweiterung der Strategien auf andere Szenarien
- Entwicklung neuer Muster zur Resilienzsteigerung
- Untersuchung ökonomischer Auswirkungen

Quellen