

Resilienz und Fehlertoleranz in verteilten Systemen

Modul „Software Engineering“

14. Januar 2025
Derhachov, Schmidt, Westholt

HTWK Leipzig, FIM

Gliederung

- 1 Resilienz und Fehlertoleranz
- 2 Strategien
- 3 Pattern und Konzepte
- 4 Fazit
- 5 Diskussion
- 6 Quellen
- 7 Cheatsheet zum Lernen

Resilienz und Fehlertoleranz

Initiales Beispiel 1.

Resilienz und Fehlertoleranz

Begriffsklärung

Begriffe

- **Resilienz**

- ▶ Funktionsfähigkeit trotz Störungen, Angriffen oder Ausfällen sowie schnelle Erholung

- **Fehlertoleranz**

- ▶ Korrektes Funktionieren trotz Fehlern oder Störungen

Resilienz und Fehlertoleranz

Begriffsklärung - Zusammenhang

Zusammenhang

- **Resilienz** = übergeordnetes Konzept, umfasst Fehlertoleranz sowie Aspekte wie Wiederherstellung, Anpassungsfähigkeit und präventive Maßnahmen
- **Fehlertoleranz** = Fokus auf unmittelbarer Bewältigung von Fehlern während des Systembetriebs

Resilienz und Fehlertoleranz

Begriffsklärung - Ursprung

Ursprung

- beide Begriffe haben ihren Ursprung nicht in der Informatik
- **Resilienz** aus dem Lateinischen *resilire*, entspricht „zurückspringen“ oder „abprallen“
- **Fehlertoleranz** aus den Ingenieurwissenschaften

Resilienz und Fehlertoleranz

Motivation

Motivation

- Störungen im laufenden Betrieb sollen verhindert werden
- Zugunsten der Sicherheit, Kundenzufriedenheit etc.

Resilienz und Fehlertoleranz

Initiales Beispiel 2.

- wie können wir Resilienz und Fehlertoleranz am Beispiel weiterführen?
- Zunächst natürlich grob, weil die Strategien und Pattern ja noch kommen.

Strategien

Resilienzstrategien

- Redundanz
- Partitionierung
- Skalierung

Strategien

Resilienzstrategien: Redundanz

Definition Redundanz

Vervielfältigung kritischer Komponenten oder Funktionen zur Erhöhung der Zuverlässigkeit und Verfügbarkeit.

- Unterschiede in Arten der Redundanz und Ebenen der Redundanz.

Strategien

Resilienzstrategien: Redundanz

Arten der Redundanz

- Aktive Redundanz
 - ▶ Mehrere Komponenten arbeiten parallel
 - ▶ Nahtloser Übergang bei Ausfall einer Komponente
- Passive Redundanz
 - ▶ Redundante Komponenten im Standby.
 - ▶ Aktivierung bei Ausfall der primären Komponente (mit Umschaltzeit)

Strategien

Resilienzstrategien: Redundanz

Ebenen der Redundanz

- Software-Redundanz
 - ▶ Mehrere Softwarekomponenten erfüllen dieselbe Funktion
- Hardware-Redundanz
 - ▶ Doppelte physische Komponenten (z. B. Netzteile, RAID-Systeme)
- Daten-Redundanz
 - ▶ Mehrfach gespeicherte Daten (z. B. Replikation, Backups)
- Netzwerk-Redundanz
 - ▶ Alternative Übertragungswege (z. B. redundante Router, Glasfaserverbindungen)
- Geografische Redundanz
 - ▶ Verteilung auf mehrere Standorte zur Minimierung großflächiger Ausfälle

Strategien

Resilienzstrategien: Partionierung

Definition Partionierung

Physische Unterteilung von Daten in kleinere, logisch zusammenhängende Einheiten für Skalierbarkeit, Leistung und Flexibilität.

- Unterschiede in Arten der Partionierung.

Strategien

Resilienzstrategien: Partionierung

Arten der Partionierung

- Horizontale Partitionierung: Aufteilung von Datensätzen basierend auf einem Partitionsschlüssel
- Vertikale Partitionierung: Gruppierung von Spalten einer Tabelle
- Funktionale Partitionierung: Organisation nach Funktion oder Zweck der Daten
- RANGE-Partitionierung: Unterteilung nach Wertebereichen (z. B. Zeit, Zahlen)
- HASH-Partitionierung: Verteilung durch Hash-Funktion für gleichmäßige Last
- Round-Robin Partitioning: Gleichmäßige, zyklische Datenverteilung

Strategien

Resilienzstrategien: Partitionierung

- Was ist das genau?
- Wie funktioniert es?
- Gerne mit Bild und/oder Code

Strategien

Resilienzstrategien: Skalierung

Definition Skalierung

Flexible Anpassung von Ressourcen an veränderte Anforderungen.

- Unterschiede in Arten der Skalierung bzw. Skalierbarkeit.

Strategien

Resilienzstrategien: Skalierung

Arten der Skalierung

- Vertikale Skalierung (Scale Up)
 - ▶ Aufrüstung von Hardware (z. B. CPU, RAM) eines Systems
 - ▶ Begrenzung auf eine zentrale Einheit
- Horizontale Skalierung (Scale Out):
 - ▶ Hinzufügen von Servern oder Instanzen
 - ▶ Verteilte Last auf mehrere Einheiten
- Automatische Skalierung:
 - ▶ Dynamische Anpassung der Ressourcen
 - ▶ Häufig in Cloud-Umgebungen für optimierte Ressourcennutzung

Strategien

Resilienzstrategien: Skalierung

- Was ist das genau?
- Wie funktioniert es?
- Gerne mit Bild und/oder Code

Strategien

Fehlertoleranzstrategien

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

Strategien

Fehlertoleranzstrategien: Fehlerbehandlung und -isolierung

- Was ist das genau?
- Wie funktioniert es?
- Gerne mit Bild und/oder Code

Strategien

Fehlertoleranzstrategien: Nutzung von Fallback-Mechanismen

- Was ist das genau?
- Wie funktioniert es?
- Gerne mit Bild und/oder Code

Strategien

Fehlertoleranzstrategien: Vermeidung kaskadierender Fehler

- hier eigene Folie überhaupt nötig?
- Verweis auf oder Übergang zu Circuit-Breaker?
- Gerne mit Bild und/oder Code

Pattern und Konzepte

Circuit-Breaker

- Isoliert fehlerhafte Dienste
- Unterbricht Anfragen bei wiederholtem Fehler
- Verhindert kaskadierende Ausfälle

Pattern und Konzepte

Circuit-Breaker: Zustandsdiagramm

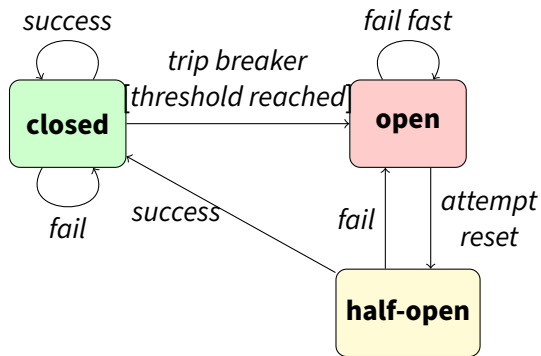


Abbildung 1: Circuit Breaker Zustandsdiagramm

Pattern und Konzepte

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.

Pattern und Konzepte

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.

Pattern und Konzepte

Circuit-Breaker: großes Diagramm mit Anpassungen

Pattern und Konzepte

Retry-Muster

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

Pattern und Konzepte

Retry-Muster: Sequenzdiagramm

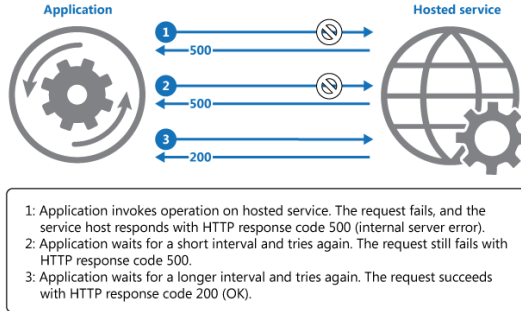


Abbildung 2: Sequenzdiagramm des Retry Patterns

Pattern und Konzepte

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.

Pattern und Konzepte

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.

Pattern und Konzepte

Kombination mit Circuit-Breaker

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

Pattern und Konzepte

Load Balancing

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

Pattern und Konzepte

Load Balancer: Architekturdiagramm

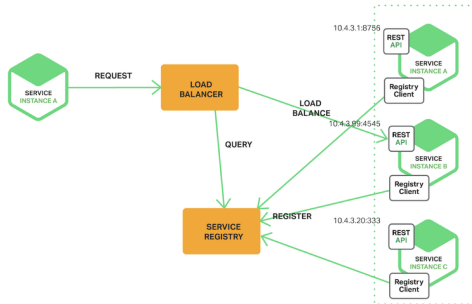


Abbildung 3: Architekturdiagramm mit einem zentralen Load Balancer

Pattern und Konzepte

Health Checks bei Load Balancing

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

Fazit

Was wurde gemacht?

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

Fazit

Fallstudie: Netflix

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

Diskussion

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

Quellen



Cheatsheet zum Lernen

- TODO, damit hätten wir etwas, was noch keine Gruppe hat.
- die wichtigsten Definitionen usw hier in kleinerer Schrift