

# **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.

Architekturdiagramm für Videostreaming-Plattform

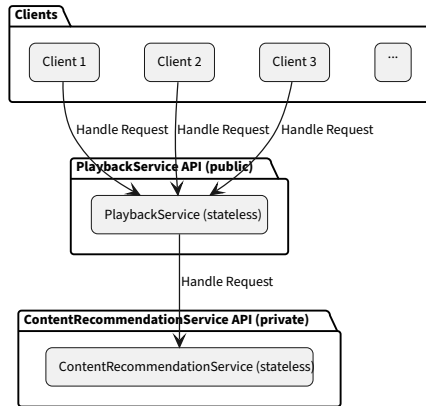


Abbildung 1: Beispiel ohne Resilienzstrategien

# 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

- 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: 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: 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.

# Strategien

## Resilienzstrategien: Arten der Partitionierung

- 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: Skalierung

### Definition Skalierung

Flexible Anpassung von Ressourcen an veränderte Anforderungen.

# Strategien

## Resilienzstrategien: 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

## Pattern für Fehlertoleranzstrategien

- Fehlerbehandlung und -isolierung mittels Retry-Pattern
- Nutzung von Fallback-Mechanismen
- Vermeidung kaskadierender Fehler mittels Circuit-Breaker

# Pattern und Konzepte

## Circuit-Breaker

### Definition - Was ist ein Circuit-Breaker?

Entwurfsmuster zur Isolation fehlerhafter Dienste in verteilten Systemen, um Überlastung zu verhindern und Stabilität zu gewährleisten.

### Aufgaben

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

## Pattern und Konzepte

### Circuit-Breaker: Zustandsdiagramm

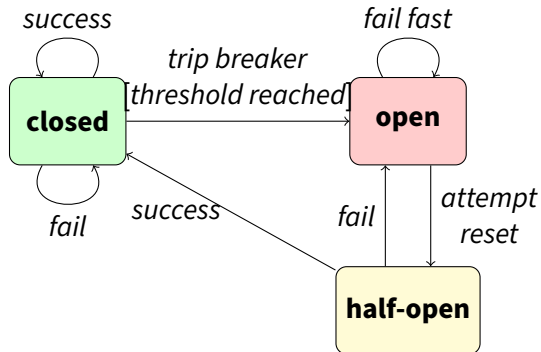


Abbildung 2: Circuit Breaker Zustandsdiagramm

# Pattern und Konzepte

## Circuit-Breaker: Implementierungsansätze

- Clientseitig
  - ▶ Abfangen externer Anfragen vor der Weiterleitung
- Dienstseitig
  - ▶ Schutz des Dienstes vor Überlastung durch viele fehlerhafte Anfragen
- Proxy-basiert
  - ▶ Circuit-Breaker zwischen Clients und Diensten in Proxys platziert

# 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

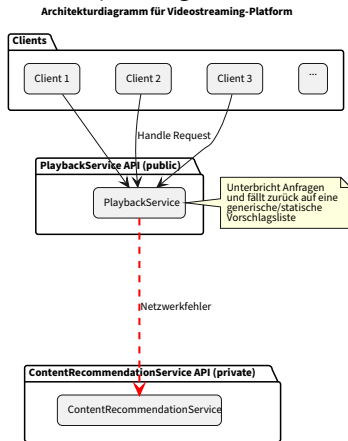


Abbildung 3: Beispiel mit Circuit Breaker

# Pattern und Konzepte

## Retry-Muster

### Definition - Was ist ein Retry-Muster?

Architekturmuster zur automatischen Wiederholung fehlgeschlagener Operationen, insbesondere bei vorübergehenden Fehlern.

### Funktionen

- Handhabt vorübergehende Fehler durch wiederholte Versuche
- Nutzt dazu exponentielle Backoff-Strategien (= Verlängerung der Wartezeit zwischen Wiederholungen)



# Pattern und Konzepte

## Retry-Muster: Sequenzdiagramm

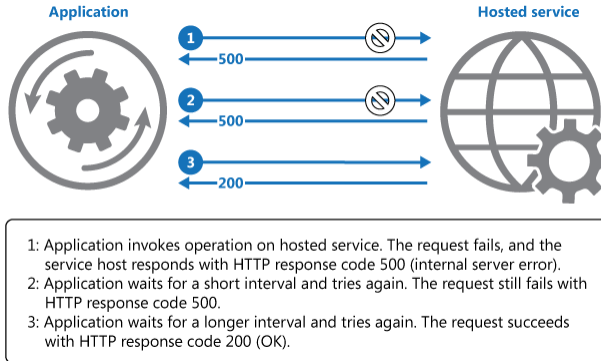


Abbildung 4: Sequenzdiagramm des Retry Patterns [3]

# 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

### Definition - Was ist Load-Balancing?

Verteilung der Last auf mehrere Server zur Verbesserung von Leistung und Ausfallsicherheit

### Arten

- Network Load Balancer
- Application Load Balancer
- Hardware Load Balancer

# Pattern und Konzepte

## Load Balancing: Service Discovery

### Service Discovery

- **Server-side Discovery**
  - ▶ Ein zentraler Load Balancer
  - ▶ Dienste sind unsichtbar für Client
- **Client-side Discovery**
  - ▶ Kein zentraler Load Balancer
  - ▶ Client sieht alle Dienstinstanzen
  - ▶ Client betreibt Load Balancing

# Pattern und Konzepte

## Load Balancing: Architekturdiagramm

### Komponenten

- Load Balancer
- Service Registry
- Dienstinstanzen

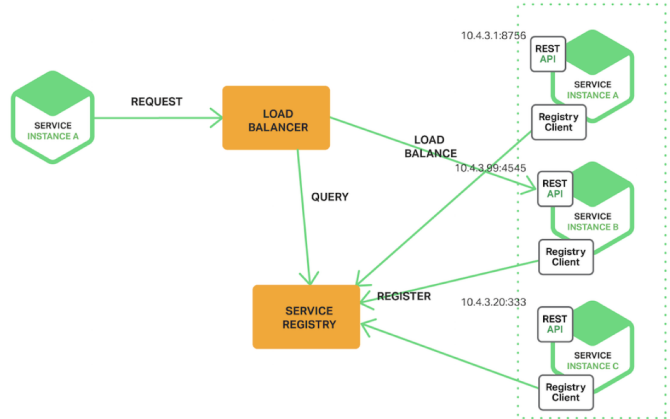


Abbildung 5: Load Balancer mit Server Side Discovery [4]

# Pattern und Konzepte

## Load Balancing: statische Strategien

### Source IP Hash Load Balancing

- Clients werden praktisch zufällig verteilt
- Anfragen eines Clients werden immer auf der selben Serviceinstanz verarbeitet (*Sticky Sessions*)

### Round Robin

- Anfragen werden zyklisch auf Instanzen verteilt



# Pattern und Konzepte

## Load Balancing: dynamische Strategien

### Least Connection

- Anstehende Aufgabe geht an den Dienst mit den momentan wenigsten aktiven Netzwerkverbindungen

### Resource Based

- Serviceknoten berichten aktuelle Auslastung (z.B. CPU-Auslastung)
- Nächste Aufgabe geht an den Knoten mit niedrigster Auslastung

# Pattern und Konzepte

## Load Balancing: dynamische Strategien

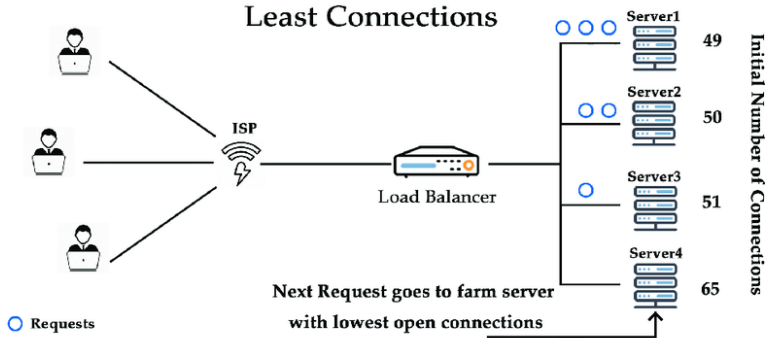


Abbildung 6: Least Connection-basiertes Load Balancing [1]

# Pattern und Konzepte

## Health Checks bei Load Balancing

### Konzept

Dienstinstanzen kommunizieren ihren Grad der Funktionsbereitschaft, um dynamische Lastverteilung zu ermöglichen.

- Dienste haben Endpunkte, um Auslastung oder Fehler abzurufen
- Überlastete Dienste erhalten weniger Anfragen
- Fehlerhafte Dienste können übergangen werden

# Pattern und Konzepte

## Load Balancing: dynamische Skalierung

### Problem

- Die Anzahl der Anfragen ist nicht immer gleich
  - ▶ Zur Spitzenbelastung sind mehr Dienstknoten nötig
  - ▶ In Ruhezeiten verschwenden unterbelastete Knoten Strom und Geld

### Die Lösung: dynamische Skalierung

- Der Load Balancer wird befähigt, Dienstinstanzen zu starten und stoppen
- Schwellwerte definieren, wann dies erfolgen soll

# Pattern und Konzepte

## Load Balancing: Vorteile

- Gesteigerter Durchsatz durch parallel laufende Dienste
- Hohe Verfügbarkeit durch Redundanz und nahtlosem Failover
- Kostenreduktion durch dynamische horizontale Skalierung

# Pattern und Konzepte

## Load Balancing: Nachteile

- Deutlich gesteigerte Komplexität der Infrastruktur
- Zusätzliche Kosten für Load Balancer (Hardware und Lizenz)
- Dienste sind nicht immer kompatibel
  - ▶ Healthchecks müssen implementiert werden
  - ▶ Stateful-Dienste sind nur eingeschränkt verwendbar
- Anstelle eines einzelnen Dienstes ist jetzt der Load Balancer ein Single Point of Failure

# Pattern und Konzepte

## Round Robin DNS

### Definition - was ist Round Robin DNS?

- Für eine Domain werden mehrere IP-Adressen hinterlegt
- DNS-Server gibt alle Adressen in rotierender Reihenfolge zurück

### Wirkung

- Externes, statisches Load Balancing
- Verfügbarkeit auch bei Totalausfall eines Servers

# Pattern und Konzepte

## Round Robin DNS

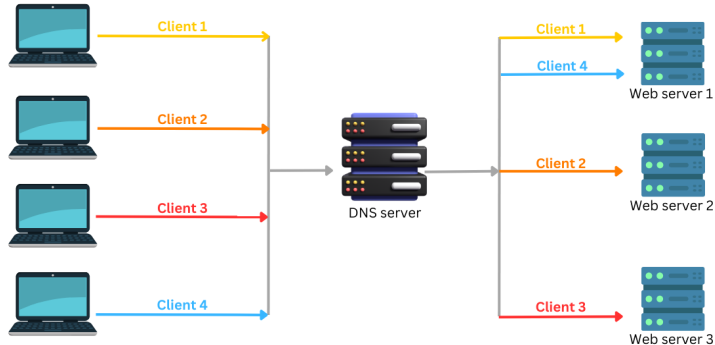


Abbildung 7: Round Robin DNS [2]



# Pattern und Konzepte

## Round Robin DNS: Vorteile

- Einfach zu implementieren
- Sticky Sessions by default
- Fallback auf eine andere IP-Adresse schützt auch bei einem Serverausfall

# Pattern und Konzepte

## Round Robin DNS: Nachteile

### Balancing

- Nur statische Lastverteilung ist möglich
- Die Reihenfolge der IPs wird vom DNS ein Mal bestimmt, aber danach gecached
  - ▶ Verteilung kann zufällig ungleich werden, da Balancing nur unregelmäßig erfolgt
  - ▶ Neue Server erst nach gewisser Zeit gleich stark ausgelastet

### Failover-Verhalten

- Vom Client kontrolliert, nicht vom Server oder DNS
  - ▶ Lange Wartezeiten bis zum Failover sind möglich

# 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 I

- [1] Bhavya Alankar, Gaurav Sharma, Harleen Kaur, Raul Valverde, and Victor Chang. Experimental setup for investigating the efficient load balancing algorithms on virtual cloud. *Sensors*, 20(24), 2020.
- [2] CloudDNS. What is Round-Robin DNS? Optimize Server Load.  
<https://www.cloudns.net/wiki/article/182/>.
- [3] Microsoft. Retry pattern.
- [4] Tom Schöner, Kai von Luck, Tim Tiedemann, Ulrike Steffens, and Stefan Sarstedt. Analyse abstrakter architekturmodelle in verteilten systemen. 2017.

# Cheatsheet zum Lernen

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

# Cheatsheet: Wichtige Definitionen und Konzepte

## 1. Grundbegriffe

**Resilienz** = Übergeordnetes Konzept, Fehlertoleranz + Aspekte wie Wiederherstellung, Anpassungsfähigkeit und präventive Maßnahmen

**Fehlertoleranz** = Fokus auf unmittelbarer Bewältigung von Fehlern während des Systembetriebs

**Ziel:** Störungen im laufenden Betrieb vermeiden

## 2. Resilienzstrategien

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

-> aktive vs. passive Redundanz; Redundanzebenen!

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

**Skalierung:** Flexible Anpassung von Ressourcen an veränderte Anforderungen.

-> vertikal vs. horizontal vs. automatisch!

## 3. Fehlertoleranzstrategien + Pattern

**Retry-Muster:** Wiederholt fehlgeschlagene Operationen, oft mit Backoff-Strategien.

**Circuit-Breaker:** Isoliert fehlerhafte Dienste, unterbricht Anfragen bei wiederholtem Fehler, verhindert kaskadierende Ausfälle.

-> clientseitig vs. dienstseitig vs. proxy-basiert!

**Load-Balancing:** Verteilung der Last auf mehrere Server zur Verbesserung von Leistung und Ausfallsicherheit.

**Load-Balancing-Strategien:**

- Round Robin DNS: rotierende Rückgabe von IP-Adressen
- Least Connection: Aufgabe an Dienst mit den wenigsten aktiven Netzwerkverbindungen