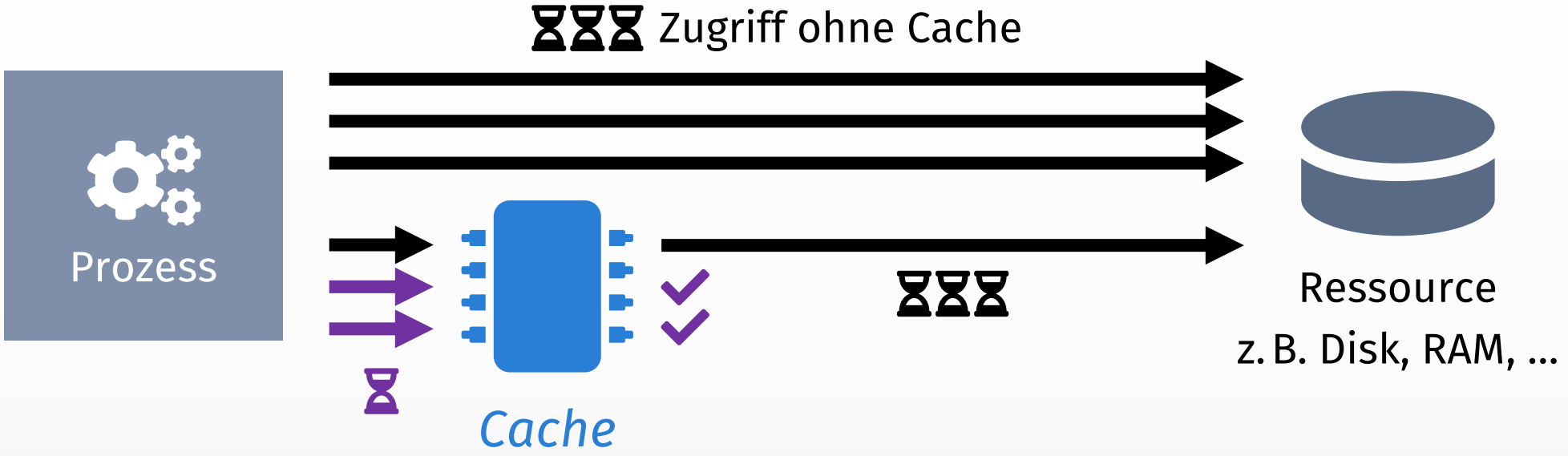
Fachhochschule Nordwestschweiz
Hochschule für Technik

Application Performance Management

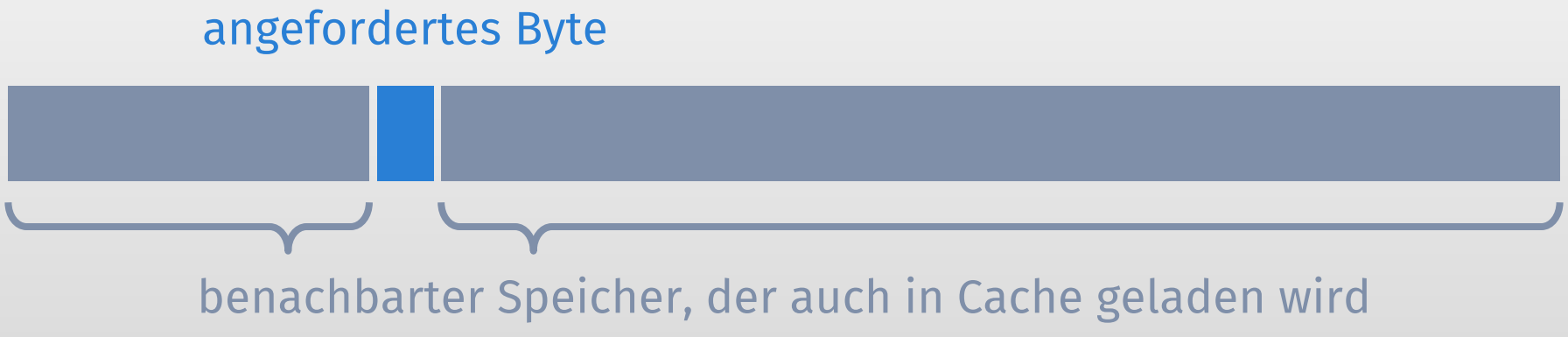
Performance Management & Autoscaling

Michael Faes

Rückblick: Caching



Konkret: CPU-Caches, DB-Caches, Caching im Web, ...



Übersicht

1. Rückblick Caching
2. Performance Management
3. Autoscaling
4. Übung
5. Prüfungsvorbereitung

Performance Management

«Management» von Performance

Bisher: Verschiedene Aspekte von Performance

Performance **messen**

Performance-**Probleme identifizieren**

Performance **charakterisieren**

Performance **vergleichen**

Performance **vorhersagen**

Performance **visualisieren**

Performance **«optimieren»**

Umsetzung in der Praxis?

- Welche Aktivitäten wann durchführen?
- Wie sicherstellen, dass Performance nicht «zum Problem» wird?

Performance Management: Methodologie um während Entwicklungszyklus sicherzustellen, dass Performance zufriedenstellend ist.

Performance berücksichtigen – wann?

Woche 1:

"Make it Work, Make it Right, *Make it Fast*."

In dieser Reihenfolge… ?

Andererseits: Kosten, um (Performance-) Probleme zu beheben, steigen!

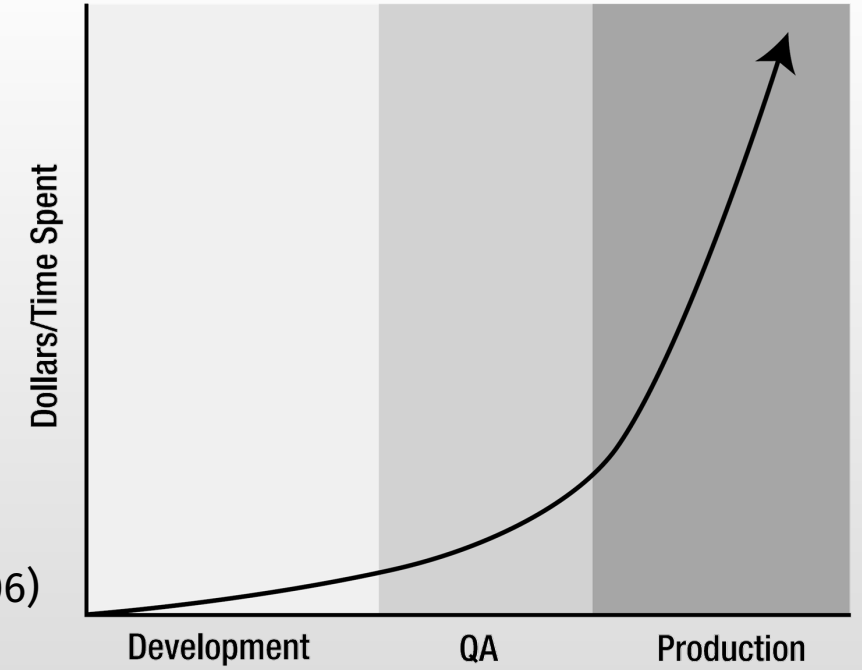


Bild: Haines (2006)

APM in der Architektur

Performance sollte bereits in der Architektur berücksichtigt werden.

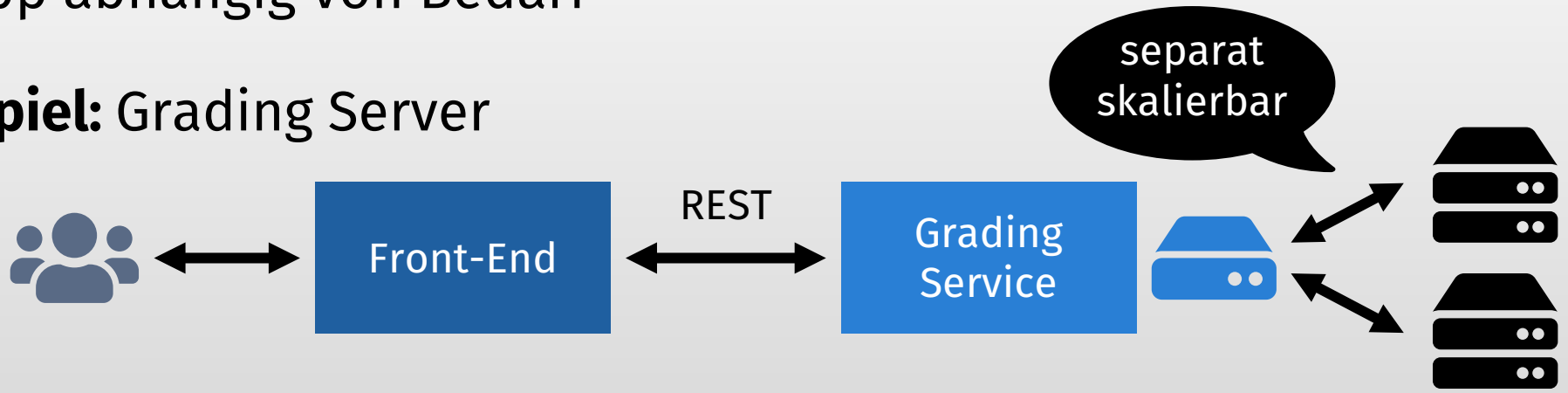
Überlegungen

1. Wichtigste Frage: Wie wird Skalierung ermöglicht/vereinfacht?

Heutzutage oft durch *REST* und *Microservices*:

- REST ermöglich Caching und vereinfacht horizontale Skalierung
- Microservices ermöglichen individuelle Skalierung von Teilen einer App abhängig von Bedarf

Beispiel: Grading Server



2. SLAs in Requirements/Use Cases

Für gewisse Anfragen ist Antwortzeit vielleicht wichtiger als für andere.

Vorteile von SLAs:

- Machen Performance-Erwartungen explizit
- Durch Diskussion entstehen realistische Erfolgskriterien
- Können später Optimierungs-Anstrengungen steuern

Beispiel:
(Haines, 2006)

USE CASE: PATIENT HISTORY SEARCH FUNCTIONALITY

Use Case
The Patient Management System must provide functionality to search for specific patient medical history information.

Scenarios
Scenario 1: The Patient Management System returns one distinct record.
Scenario 2: The Patient Management System returns more than one match.
Scenario 3: The Patient Management System does not find any users meeting the specified criteria.

●●●

SLAs
Scenario 1: The Patient Management System will return a specific patient matching the specified criteria in less than three seconds for 95 percent of requests. The response time will at no point stray more than two standard deviations from the mean.
Scenario 2: The Patient Management System will return a collection of patients matching the specified criteria in less than five seconds for 95 percent of requests. The response time will at no point stray more than two

APM während Entwicklung

Umfangreiches Unit-Testing ist Grundanforderung für erfolgreiche Software-Entwicklung. Aber oft nur funktionale Tests!

Wenn möglich, sollten auch *Performance Unit Tests* geschrieben werden. Fokus auf wichtige Bereiche. ➞ SLAs!

Herausforderungen

- Performance-Tests schwieriger auszuwerten als funktionale; Performance ist variabel und nicht unbedingt deterministisch
- Unit-Tests sollten schnell sein, aber Performance-Tests sollten genügend «Zeit haben» um Performance zuverlässig zu evaluieren

Möglichkeiten/Überlegungen

1. Benchmarks für wichtige Codestücke schreiben. → *JMH*
   Können zu definierten Zeitpunkten von Hand ausgeführt werden oder automatisiert. Automatische Pass/Fail-Auswertung schwierig, aber gibt zumindest Daten.
2. Relative Auswertung möglicherweise einfacher.
   Beispiel: «*Mit 10× mehr Daten soll Codestück höchstens 12× so lange dauern.*»
3. Coverage für Performance-Tests erheben. Gibt Sicherheit, dass wichtige Teile des Codes nicht untergehen.
4. Profiling von Speicher- und Zeitverbrauch von wichtigen Codestücken. Kann auch automatisiert werden. → *JMH*

APM während QA

Code sollte QA nur bestehen, wenn auch Performance-Anforderungen bestanden werden. ➞ SLAs!

Pendant zu Integrations-/System-Tests: Endbenutzer-Antwortzeiten messen. ➞ Load Testing (z. B. mit *JMeter*)

- Auch hier Automatisierung möglich!

Herausforderung: Antwortzeit alleine reicht nicht. System muss unter angenommener oder definierter Last und mit realistischer Hardware getestet werden!

- Möglichkeit, Last der Hardware entsprechend herunterzuskalieren

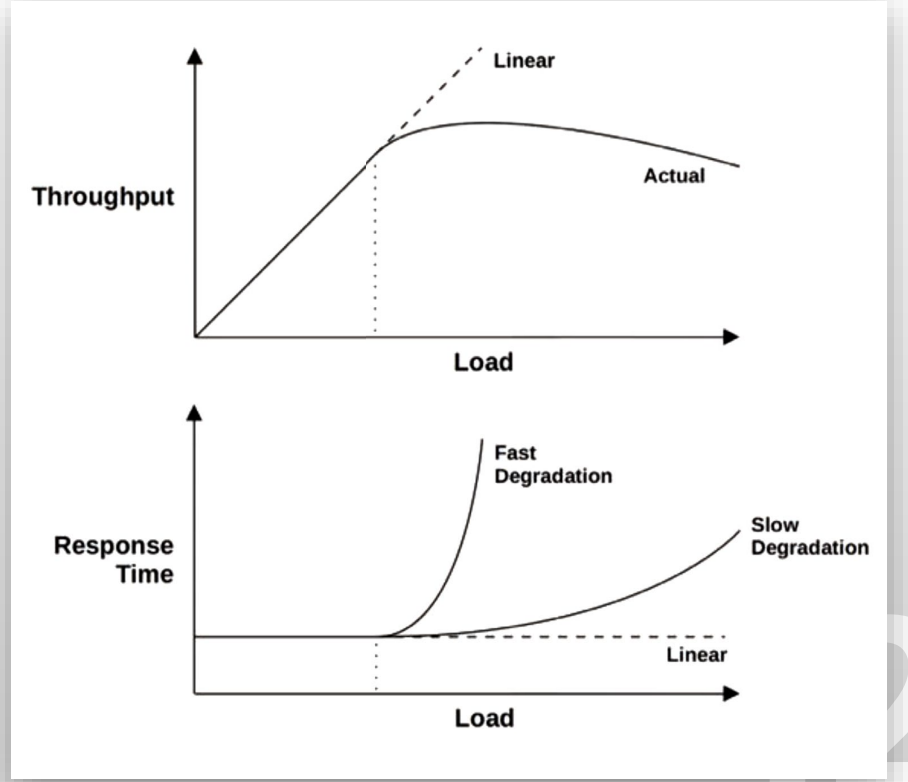
APM während Preproduction

Während Preproduction (Staging) steht oft realistische Umgebung/Hardware bereit, auf welcher aussagekräftige Performance-Analysen gemacht werden können.

Gelegenheit, um Kapazität der App zu testen → Stress Testing (*JMeter*)

Mögliche Fragen

- Wie «gut» (knapp) werden SLAs unter erwarteter Last erfüllt?
- Ab welcher Last werden SLAs nicht mehr erfüllt?
- *Welche* SLAs werden als erstes nicht mehr erfüllt?



Quelle: Gregg (2020)

APM während Production

Performance sollte laufend gemessen und evaluiert werden.

Mögliche Fragen

- Wie hoch ist der Ressourcenverbrauch? Welche Ressourcen sind am stärksten ausgelastet?
- Wie gross sind die Endbenutzer-Antwortzeiten tatsächlich?
- Was machen die User in der Realität?
  Falls Verhalten deutlich anders als erwartet: Testpläne anpassen, in Preproduction neu durchführen

Autoscaling

Autoscaling

Autoscaling: Automatisches Hinzufügen/Entfernen von Instanzen je nach Last

Scale-out/Scale-in

Monitoring von Performance-Metriken

Berechnen von *Performance-Indikatoren*

PI-Definitionen

Vergleich mit *Policies* (Schwellwerte o. ä.)

Policy-Definitionen

Performance-Indikatoren

Performance-Indikatoren sind Grundlage für *Scaling-Policies*.

Können einfache Metriken sein

- Speicherverbrauch, CPU-Last, Antwortzeit (In-Server), …

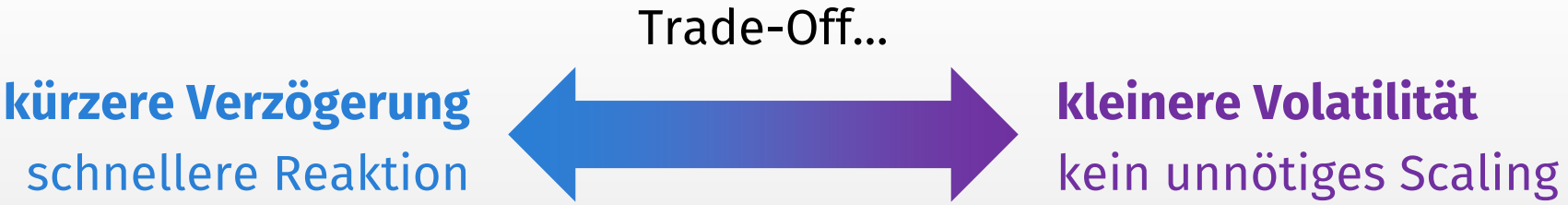
Oder higher-level Indikatoren

- Kombination mehrerer Metriken oder externe Metriken
- Aggregierung über definierten Zeitraum (Ø, Median, …)
- Beispiele: EU-Antwortzeit, aktive User, Kosten, …

Kriterien für gute Indikatoren

Was sind *Kriterien* für gute Performance-Indikatoren für Autoscaling?

1. Periodische, automatische Erfassung möglich
2. Kleine Verzögerungszeit
3. Kleine Volatilität



4. Aussagekraft für Massnahmen
5. Verständlichkeit (für Policy-Definitionen)

Policies: Schwellwerte

Zwei Arten von Policies:

- *Schwellwert-basiert*: Policy besteht aus Regeln. Jede Regel besteht aus Indikator, Schwellwert und Anpassung, z. B. «+1 Instanz»
- *Zielwert-basiert*: Policy besteht aus Indikator und Zielwert. Instanzen werden automatisch hinzugefügt und entfernt, so dass Zielwert erreicht wird.

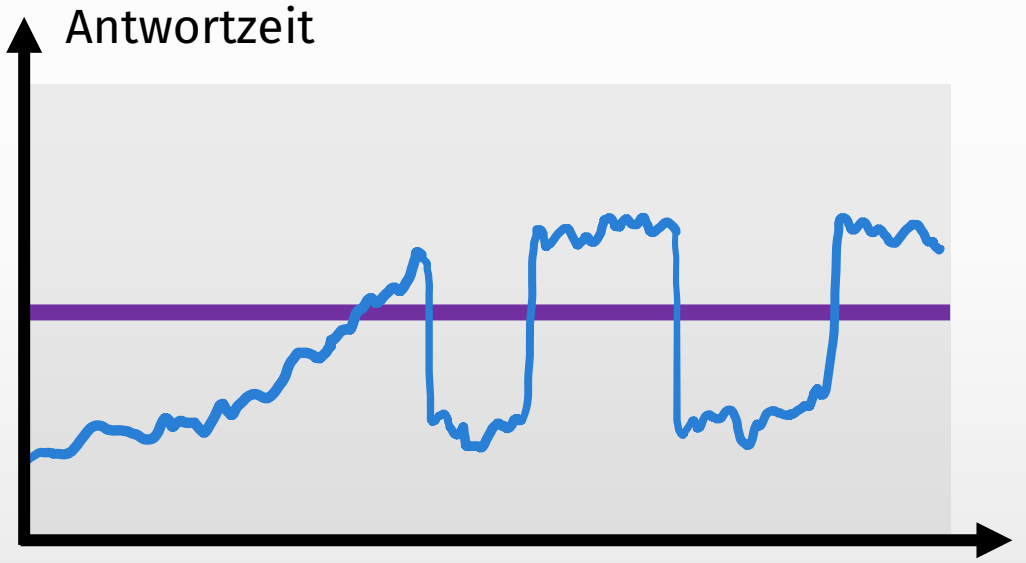
Beispiel Schwellwert-basiert:

1. Wenn CPU-Auslastung *über 90%* wächst, füge 1 neue Instanz hinzu
2. Wenn CPU-Auslastung *unter 90%* fällt nimm 1 Instanz weg

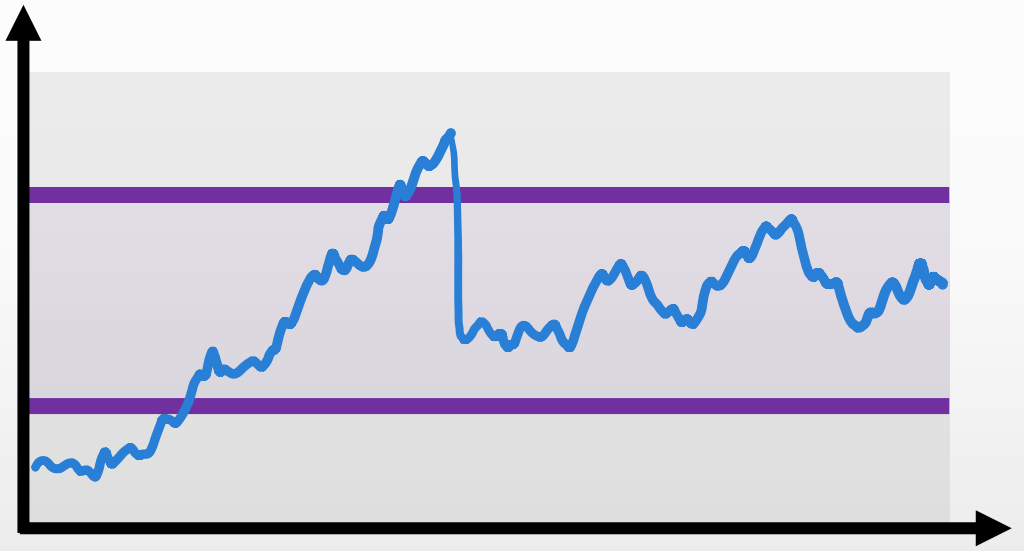
Problem mit diesen Regeln?

Hysterese

Hysterese: Verzögerte Änderung der Wirkung einer Ursache.
In diesem Fall: Überschreiten eines Schwellwerts ➞ Scaling verzögert



Ohne Hysterese



Mit Hysterese

Hier: Zwei unterschiedliche Schwellwerte. Auch möglich: zeitliche Verzögerung

Beispiel Schwellwerte: App Auto-Scaler

```
"scaling_rules": [{
    "metric_type": "memoryutil",
    "breach_duration_secs": 600,
    "threshold": 90,
    "operator": ">=",
    "cool_down_secs": 300,
    "adjustment": "+1"
}, {
    "metric_type": "memoryutil",
    "breach_duration_secs": 600,
    "threshold": 30,
    "operator": "<",
    "cool_down_secs": 300,
    "adjustment": "-1"
}]
```

memoryused,
memoryutil,
cpu,
responsetime,
throughput

github.com/cloudfoundry/app-autoscaler-release/blob/main/README.md

«Finde die Probleme»

```
"scaling_rules": [{
    "metric_type": "cpu",
    "breach_duration_secs": 180,
    "threshold": 100,
    "operator": ">",
    "cool_down_secs": 20,
    "adjustment": "+1"
}, {
    "metric_type": "cpu",
    "breach_duration_secs": 180,
    "threshold": 5,
    "operator": "<=",
    "cool_down_secs": 20,
    "adjustment": "-1"
}]
```

1

```
"scaling_rules": [{
    "metric_type": "memoryutil",
    "breach_duration_secs": 10,
    "threshold": 60,
    "operator": ">=",
    "cool_down_secs": 300,
    "adjustment": "+1"
}, {
    "metric_type": "memoryutil",
    "breach_duration_secs": 60,
    "threshold": 20,
    "operator": "<=",
    "cool_down_secs": 300,
    "adjustment": "-1"
}]
```

2

Policies: Zielwerte

Alternative zu Schwellwerte: *Zielwerte*

Beispiel Kubernetes: Konfiguration für *Horizontal Pod Autoscaler*

```
minReplicas: 1
maxReplicas: 10
metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 60
```

Vorteile: Einfacher, verständlicher, weniger fehleranfällig

Nachteil: Weniger Flexibilität

Fragen?