

Architektur für skalierbare Webanwendungen

Vladislav Faerman

Hochschule für angewandte Wissenschaften München
Fakultät für Informatik und Mathematik

30. Mai 2017

Agenda

Einleitung

Motivation und Ziel

Theorie

Skalierbarkeit

Wartbarkeit

Relevante Technologien

Architektur

Konzept

Prototyp

Fazit



Motivation und Ziel

Architektur für Webanwendungen aufstellen, die die Entwicklung von skalierbaren, wartungs- und erweiterungsfähigen Webanwendungen ermöglicht.

Agenda

Einleitung

Motivation und Ziel

Theorie

Skalierbarkeit

Wartbarkeit

Relevante Technologien

Architektur

Konzept

Prototyp

Fazit

Skalierbarkeit

Skalierbarkeit

Die Fähigkeit eines Systems, aufgrund der wachsenden Anforderungen, entweder die Leistung der vorhandenen Ressourcen zu verbessern oder zusätzlich die neuen Ressourcen hinzufügen.

Bei der Skalierbarkeit sind zwei Arten zu unterscheiden, eine *vertikale* und eine *horizontale* Skalierbarkeit.

Skalierbarkeit

Vertikale Skalierbarkeit

Anstreben einer qualitativen Steigerung der Leistungsfähigkeit der bereits eingesetzten Ressourcen.



Skalierbarkeit

Horizontale Skalierbarkeit

Im Gegensatz zur vertikalen Skalierbarkeit wird bei der horizontalen Skalierbarkeit die Last auf zusätzliche Rechner verteilt.

ACID-Prinzip

■ Atomicity (Atomarität)

- ▶ die Transaktion wird entweder ganz oder gar nicht ausgeführt

■ Consistency (Konsistenz)

- ▶ vor und auch nach dem Ablauf einer Transaktion werden die Integrität und Plausibilität der Datenbestände gewährleistet

■ Isolation (Isolation)

- ▶ um unerwünschte Nebenwirkungen zu vermeiden, werden die Transaktionen gekapselt

■ Durability (Dauerhaftigkeit)

- ▶ gewährleistet nach einer erfolgreichen Transaktion die Persistenz aller Datenänderungen

ACID-Prinzip: Problem in verteilten Datenbanken

In verteilten Datenbanken kommt es zu Problemen, wenn alle ACID-Eigenschaften erfüllt werden sollen.

Skalierbarkeit

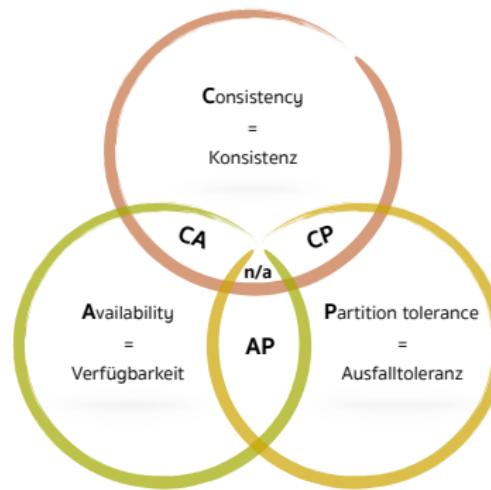
Das CAP-Prinzip

Im Jahr 2000 präsentierte Eric A. Brewer das CAP-Theorem - ein Ergebnis seiner Forschungen zu verteilten Systemen:

- **Consistency (Konsistenz)**
 - ▶ Korrektheit der in der Datenbank gespeicherten Daten
- **Availability (Hochverfügbarkeit)**
 - ▶ Alle Anfragen an das System werden mit akzeptabler Reaktionszeit stets beantwortet
- **Partition Tolerance (Partitionstoleranz)**
 - ▶ Die Ausfalltoleranz eines Knotens bzw. eines Servers aus einem Cluster

Skalierbarkeit

Das CAP-Prinzip



Die drei Anforderungen sind laut Brewer gleichzeitig nicht zu erfüllen.

Das CAP-Prinzip

Die Anforderungen in Paaren klassifizieren bestimmte Datenbanktechnologien.

- **CA (Consistency und Availability)**
 - ▶ für die klassischen relationalen Datenbankmanagementsysteme (RDBMS)
- **CP (Consistency und Partition tolerance)**
 - ▶ z. B. für Banking-Anwendungen
- **AP (Availability und Partition tolerance)**
 - ▶ die Social-Media-Sites wie Twitter oder Facebook



Skalierbarkeit

Das BASE-Prinzip

Folgendes Prinzip beschreibt eine Alternative zu den strengen ACID-Kriterien und wird im Umfeld der NoSQL-Datenbanken verfolgt:

- **Basically Available**
- **Soft State**
- **Eventually Consistent**

Bei den Systemen, die nach dem BASE-Prinzip gestaltet sind, wird bewusst in Kauf genommen, dass die Daten nach Schreiboperationen eine absehbare Zeit inkonsistent sein können.



Wartbarkeit

Die SOLID-Prinzipien

Fünf Prinzipien des objektorientierten Designs:

- Single responsibility principle
- Open/closed principle
- Liskov substitution principle
- Interface-segregation principle
- Dependency inversion principle

Bei korrekter Anwendung dieser Prinzipien erfolgt eine höhere Wartbarkeit am Softwareprodukt.

Single responsibility principle (SRP)

Für die Änderung einer Klasse kann nur einen Grund geben.



Open/closed principle (OCP)

Die Klassen/Module sollen für die Erweiterung offen sein, die bestehenden Klassen sollen jedoch nicht geändert werden.



Liskov substitution principle (LSP)

Die Subklassen dürfen das Verhalten der Elternklassen nicht ändern. Der Code, der auf bestehenden Funktionen der Elternklassen aufgebaut ist, muss auch mit Subklassen fehlerfrei funktionieren.



Interface-segregation principle (ISP)

Die Interfaces sollen so klein wie möglich sein und nur einzelne Funktionen abdecken.



Dependency inversion principle (DIP)

Die Abhängigkeiten zwischen Modulen sollen über Abstraktionen (Interfaces) gekoppelt werden. Ein Modul soll eine direkte Abhängigkeit zu den anderen Modulen vermeiden, die Abhängigkeiten werden zu den Interfaces definiert.



Dependency Injection (DI)

Dependency Injection ist der nächste Schritt nach Dependency Inversion. Das Dependency Injection Pattern basiert auf dem Inversion of Control Konzept.

- Die Art, wie die Abhängigkeiten zwischen Objekten verwaltet werden
- **Ziel:** Anwendungsobjekte voneinander entkoppelt zu halten -> **lose Kopplung**

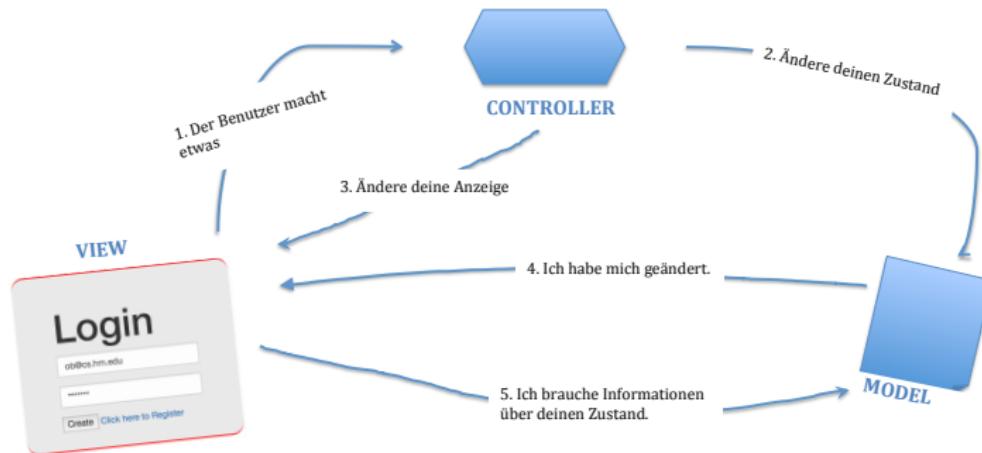


MVC-Pattern

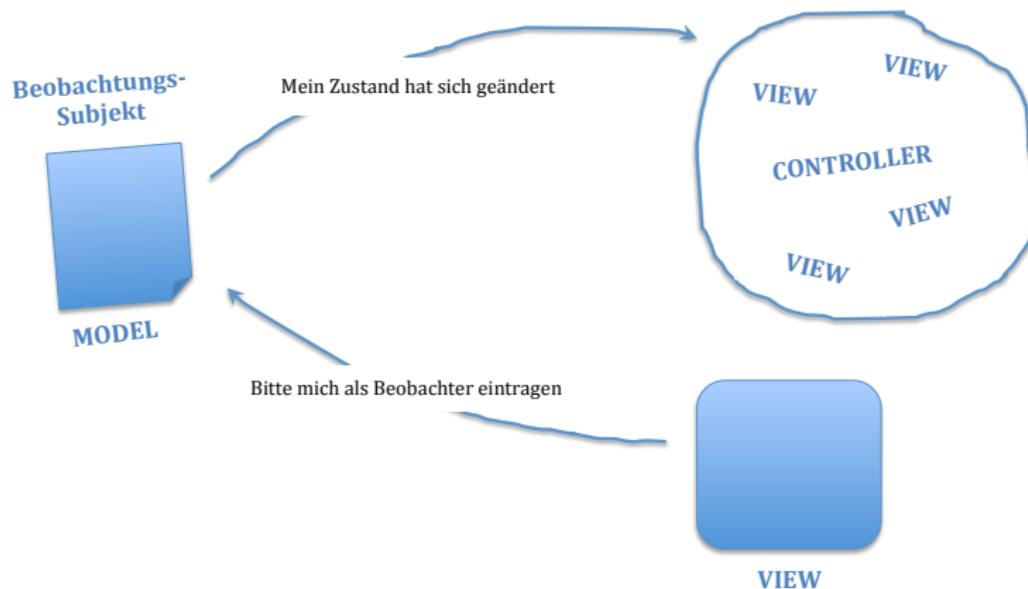
MVC ist ein Prinzip der modernen Programmierung und ist nach wie vor das wichtigste und verbreitetste Muster für die Architektur von GUI-Anwendungen.

Ziel: die Geschäftslogik einer Anwendung von der Benutzerschnittstelle trennen, was eine spätere Änderung oder Erweiterung erleichtert und eine Wiederverwendbarkeit und Austauschbarkeit einzelner Komponenten ermöglicht.

Workflow zum MVC-Konzept



Observer Pattern





Relevante Technologien

NoSQL-Datenbanken

NoSQL -Datenbanken sind nicht relationale Datenbanken, die eine Vielzahl von Datenmodellen verwenden. Unter Anderem sind es Dokumenten, Diagrammen, Schlüsselwerten und Spalten.



Relevante Technologien

MongoDB

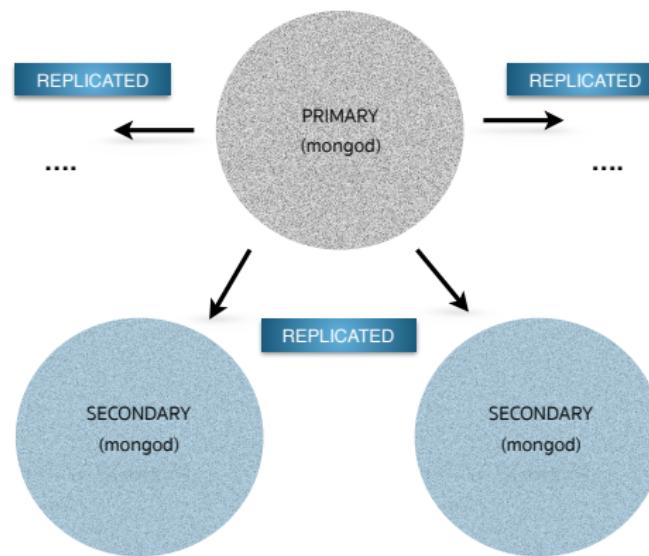
MongoDB ist eine von vielen NoSQL-Datenbanken, die als eine schemalose, dokumentenorientierte Open-Source-Datenbank bekannt ist.

Wichtige Konzepte sind **Ausfallsicherheit** und **horizontale Skalierung**.



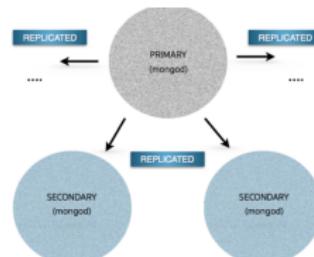
Relevante Technologien

MongoDB: Replikation (Replication)

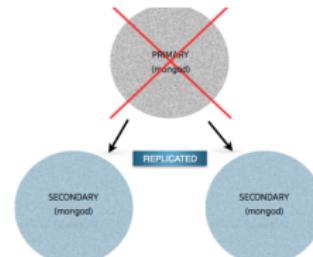


Relevante Technologien

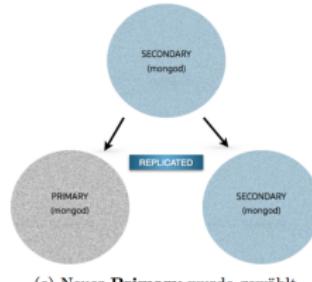
Szenario für eine Replikationsgruppe mit drei Servern in einer Shard



(a) Initialer Zustand des Replica Sets



(b) Ausfall des Primary-Knotens

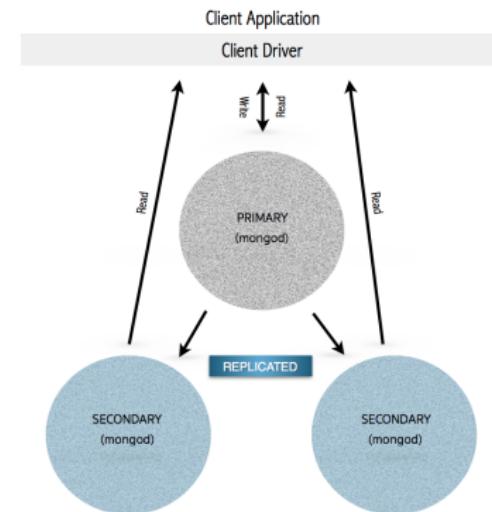
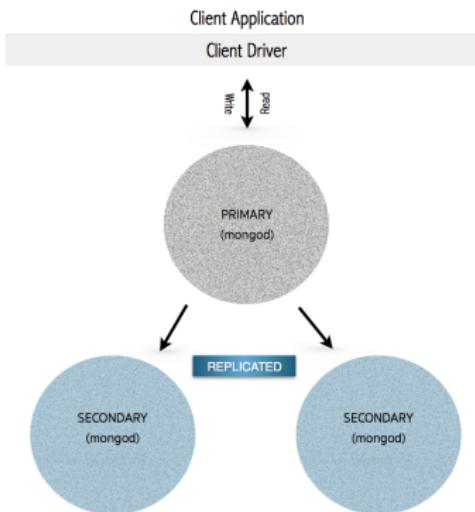


(c) Neuer Primary wurde gewählt



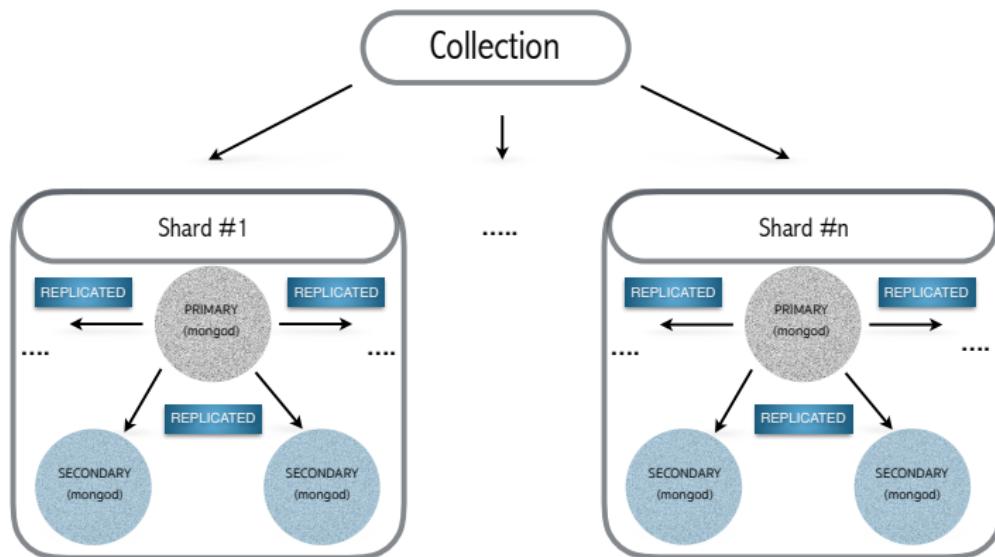
Relevante Technologien

Freischaltung der Lesezugriffe



Relevante Technologien

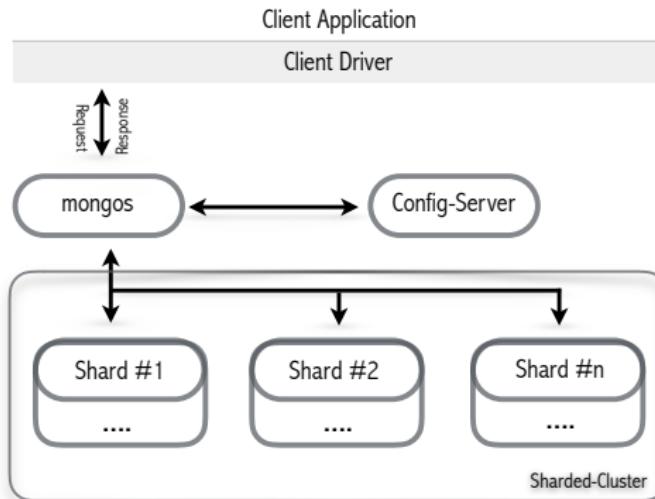
MongoDB: Horizontale Skalierung (Sharding)





Relevante Technologien

MongoDB: Sharded Cluster





Relevante Technologien

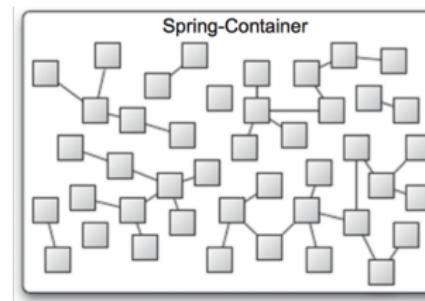
Spring Framework (1)

- Open Source Java Framework
- fundamentale Mission:
 - ▶ die Entwicklung mit Java/Java EE zu vereinfachen
- zwei wichtige Kernstrategien:
 - ▶ lose Kopplung durch „Injizieren“ von Abhängigkeiten und Interface-Orientierung
 - ▶ Reduzierung von Boilerplate-Code durch Vorlagen

Relevante Technologien

Spring Framework (2)

- Verwaltung von einfachen Java Objekten, sogenannte Plain-Old-Java-Objects (POJO) durch Spring Framework als **Spring-Beans**.



- Spring nutzt POJOs bzw. Java-Beans, indem sie per DI zusammengesetzt werden.



Relevante Technologien

REpresentational State Transfer (REST)

REST ist ein Designkonzept für Web Services. Die Daten werden in der Form von Ressourceneinheiten ausgetauscht. Jede Ressource ist eindeutig und mit einer URI identifizierbar.

- GET - ist ein lesender Zugriff auf Daten.
- PUT - aktualisiert bestehende Daten.
- DELETE - löscht vorhandene Daten.
- POST - legt neue Daten an.

Die Rückmeldung des Web-Servers erfolgt im JSON-Format.

Relevante Technologien

AngularJS 2 - JavaScript Framework

AngularJS dient zur Entwicklung von so genannten Single-page-Anwendungen.

In einer Single-page Webanwendung werden die HTML-Seite mit dem ganzen Inhalt nur einmal geladen und die Teile davon dynamisch nachgeladen oder upgedated.

Entwickelt in TypeScript. TypeScript ist eine JavaScript-Erweiterung, die durch die Benutzung von Interfaces, Klassen, Modulen und Vererbung eine typisierte und klassenbasierte JavaScript-Programmierung ermöglicht.

Agenda

Einleitung

Motivation und Ziel

Theorie

Skalierbarkeit

Wartbarkeit

Relevante Technologien

Architektur

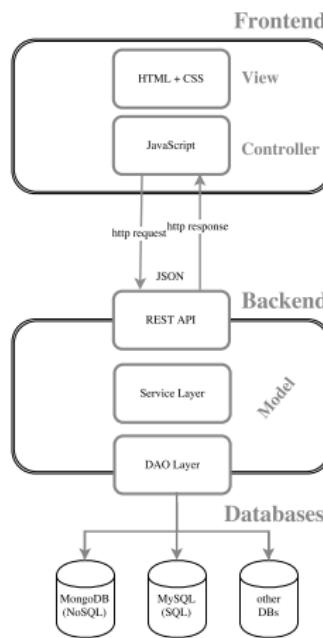
Konzept

Prototyp

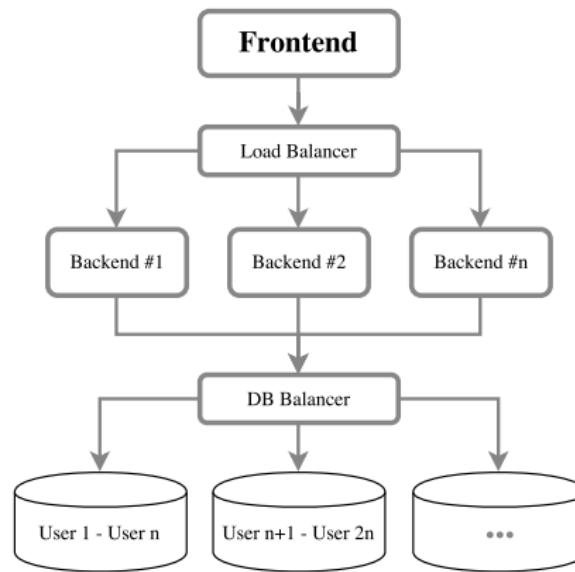
Fazit

Architektur

Konzept



Konzept: Backend



Agenda

Einleitung

Motivation und Ziel

Theorie

Skalierbarkeit

Wartbarkeit

Relevante Technologien

Architektur

Konzept

Prototyp

Fazit

!!! Live !!!



Agenda

Einleitung

Motivation und Ziel

Theorie

Skalierbarkeit

Wartbarkeit

Relevante Technologien

Architektur

Konzept

Prototyp

Fazit

Fazit

- Horizontale Skalierung - die bessere Wahl für verteilte Systeme
- *stateless*- zustandslose Client-Anfragen ermöglichen n-Anzahl von Backends, die die Anfragen unabhängig voneinander parallel und effizient bearbeiten
- bla

